AFIT/GCS/ENG/93-10

WEAPON SYSTEM INTEGRATION FOR THE AFIT VIRTUAL COCKPIT

THESIS

William Edward Gerhard, Jr.
Captain, USAF

AFIT/GCS/ENG/93-10

Acknowledgements

I am dedicating this thesis to my father. The [illegible] I learned from him were invaluable.

My thanks goes out to Lt Col Phil Amburn and his daughters who persuaded him to stay and see this project through. I would also like to thank Lt Col David Neyland, our sponsor at the Advanced Research Projects Agency.

I must acknowledge the contributions of the other graphics students, Matt Erichsen, Andrea Kunz, Mark Snyder, Mike Gardner, Brian Soltz, Alain Jones, and Kirk Wilson. I have never worked with a more capable group of people. I will fondly remember the long hours we spent in the lab together; freezing, boiling, pouring over lines of code, yelling at each other, and most important of all, laughing.

Most of all I would like to thank my wife, Nancy. Your support kept me going when nothing else could. Thank you for always being there when I needed you.

William Edward Gerhard, Jr.

Table of Contents

		Page
Acknowledgements		ii
List of Figures		v
List of Tables		vi
Abstract		vii
I.	Introduction	1
	Overview	1
	Problem Statement	4
	Scope	5
	Approach and Methodology	6
	Hardware/Software	8
	Thesis Overview	9
II.	Background	10
	Overview	10
	Flight Simulators	10
	Virtual Environments	16
	Fidelity versus Rendering	19
	Distributed Interactive Simulation	20
III.	System Design	25
	Overview	25
	Performer	25
	ObjectSim	29

Page

Utilizing Other Processes 31

Improving Appearance and Adding Detail 34

Sound 38

IV. Weapon Controller 40

M-61A-1 CANNON 44

Mk-82, Mk-83, and Mk-84 Bombs 47

GBU-12, GBU-16, and GBU-10 Bombs 49

GBU-15 50

Missiles 51

AIM-120 51

AIM-9 52

AIM-7 53

AGM-65 54

V. Head Up Display 55

Cues 58

Viewing Transformation 59

VI. Results and Recommendations 62

Results 62

Recommendations 65

Bibliography 67

Vita 70

List of Figures

Figure Page

1. Antoinette trainer 12
2. Electronic Analog Simulator 13
3. Flight Simulator with Computer Generated Image on a Dome 15
4. DIS Entity Coordinate System 21
5. Geocentric Cartesian Coordinate System 22
6. Nodes in the IRIS Performer Database 26
7. Performer Application/Cull/Draw Pipeline 28
8. ObjectSim Class Relations 30
9. Virtual Cockpit v1.0 Layout 34
10. Instrument Descriptions in Performer Tree 36
11. Actual F-15E Cockpit Layout 37
12. Virtual Cockpit Layout 38
13. Weapons Configuration File 41
14. Weapons loaded on the wings 43
15. Alignment of Body Axis During Flight 48
16. Virtual Cockpit Head Up Display 56
17. Side View of Virtual Cockpit Head Up Display 60

List of Tables

Table		Page
1.	Quality of Illusion	19
2.	Application Process Execution Times (in ms)	33
3.	Virtual Cockpit Munitions	40
4.	Virtual Cockpit Capabilities - Past, Present, and Future	63

AFIT/GCS/ENG/93-10

Abstract

The Air Force Institute of Technology is continuing research in the Virtual Cockpit. The Virtual Cockpit makes use of high performance graphics workstations, Virtual Environment technology, and Distributed Interactive Simulation network protocols to create a flight simulator based on the capabilities of the McDonnell Douglas F-15E Strike Eagle. The work presented in this thesis focuses on the design and implementation issues for integrating a weapons delivery capability. Weapons simulated include: RADAR and IR guided air-to-air missiles, gravity and precision guided bombs, and a 20mm cannon. Virtual Environment displays used include: color NTSC and monochrome high resolution helmet mounted displays employing a Polhemus Fastrack sensor, and a display using five separate BARCO projectors simulataneously. The Target graphics system was a four processor, SGI Onyx workstation with a Reality Engine graphics pipeline. Graphics rendering was accomplished with an AFIT developed object oriented simulation software package based on the SGI Performer 1.2 application development environment.

WEAPON SYSTEM INTEGRATION FOR THE AFIT VIRTUAL COCKPIT

I. Introduction

Overview

> No general can accustom an army to war. Peacetime maneuvers are a feeble substitute for the real thing; but even they can give an army an advantage over others whose training is confined to routine, mechanical drill. To plan maneuvers so that some of the elements of friction are involved, which will train officers' judgment, common sense, and resolution is far more worthwhile than inexperienced people might think. It is immensely important that no soldier, whatever his rank, should wait for war to expose him to those aspects of active service that amaze and confuse him when he first comes across them. If he has met them even once before, they will begin to be familiar to him. - Carl von Clausewitz (7:122)

Over 150 years ago General Carl von Clausewitz recognized the benefits of training soldiers under realistic conditions. While no substitute for actual combat experience exists, peacetime maneuvers involving large numbers of personnel, such as REFORGER and Red Flag, provide the most realistic training possible. In addition to training under battle field conditions, REFORGER exercises had the extra benefit of training over the terrain on which a central European battle might actually have occurred. However, the cost of moving troops to and from the maneuver and the material expended make field exercises an expensive proposition.

As one method of reducing training costs, the United States Air Force uses interactive simulators to augment pilot instruction by providing procedural training. In procedural training, the student concentrates on how to do something, in this case, how to fly an airplane (2:43). The use of flight simulators reduces training costs by conserving fuel and reducing wear on airframes. Additional benefits are increased safety and a decreased environmental impact. Nothing is damaged, except a pilot's pride, if a simulator crashes.

The expense of traditional interactive simulators prevent them from being used more widely. Although training in simulators is cheaper and safer than using actual aircraft, the cost of an operational flight simulator can exceed $20 million (23:196) (25:154). The highest fidelity interactive simulators require a large curved projection screen to create the out-the-window cockpit view. The simulator projects computer generated images onto the screen which typically has a radius of 20 feet (23:153). These hardware requirements usually dictate dedicated buildings and support staff. Due to these requirements, a dome-based simulator is not portable, and each simulator runs scenarios for only a single type of weapon system.

Research into computer technology has the potential to overcome some of the size and cost problems of traditional simulators and provide some additional benefits. It may now be possible to replace the projection screen with Virtual Environment (VE) technology. Used in this way, VE technologies could possibly reduce the size and cost of a simulator. An additional advantage of new computer technology would be electronically connecting several simulators together. This would allow pilots to fly against each other and might be a way of providing some of the same training benefits as field exercises.

The development of high speed computer networks provides a way of linking several interactive simulators together. A simulator broadcasts its position, orientation, and status on to the network. Long-haul networks provide communications between simulators that may be located anywhere in the country. The personnel training in the linked simulators interact in the same simulated environment. Ideally, the views are consistent for everyone, and the actions of one person have a definite impact on the other simulators sharing the environment. The use of networked simulators allow platoons, companies, and even battalions to fight force-on-force engagements without leaving the garrison (20:577).

The long-haul network connections allow air defense, attack helicopter, and close air support flight simulators to interact in the same environment. The battlefield is no longer confined to a single simulator; it has become distributed over several simulators. Simulators used in this manner provide many of the same benefits as the large scale field exercises, without incurring the cost. The Army has pioneered the networking of simulators in a project called SIMNET. Over 60 M-1 Abrams tank simulators at Fort Knox are linked together allowing the crews of each simulator to fight in the same simulation environment. Apache helicopter simulators at Fort Rucker are also connected to SIMNET, enabling combined arms training.

The Advanced Research Project Agency (ARPA) also uses networked simulators extensively in the WAR BREAKER project. The goal of the WAR BREAKER project is to create an end-to-end system that detects, identifies, targets, and neutralizes time-critical targets (22:2). The impetus behind WAR BREAKER was the hunt for the SCUD missiles during Operation Desert Storm. Prior to Operation Desert Storm, there was no doctrine on how to employ resources to eliminate time-critical targets. Because of the quick pace of the war, the ad-hoc methods created to destroy these targets were not fully tested or evaluated. As a way of evaluating future doctrine, proposed hardware components, and training methods, ARPA is creating a simulation environment. This simulation environment combines simulators from all branches of the Armed Forces communicating over a network called the Defense Simulation Internet. In addition to testing new tactics or proposed weapons, this simulation environment serves as a prototype for future distributed interactive simulations.

To facilitate the passing of information between a wider array of simulators, the Institute for Simulation and Training has developed a proposed IEEE standard protocol for Distributed Interactive Simulation or DIS (14). In a distributed simulation, there is no central computer arbitrating among the various players. Each participant is responsible for keeping track of what is happening in a simulation, better known as ground truth. The perception of events or other players is determined by the receiving simulator. This architecture will theoretically allow a large number of participants to interact in the same simulation.

The Air Force Institute of Technology is supporting the WAR BREAKER project by creating a flight simulator based on McDonnell-Douglas F-15E, Strike Eagle. This simulator is a multi-year project and is currently known as the AFIT Virtual Cockpit. The Virtual Cockpit is designed to run on a Silicon Graphics workstation and uses virtual environment technology to provide the out-the-window view. The cost of the workstation and supporting equipment is approximately a tenth of the cost of a traditional dome simulator (34:20). The Virtual Cockpit is also capable of broadcasting and receiving information over a computer network. At the end of last year the Virtual Cockpit adhered to the SIMNET protocol and could fly with other simulators in a WAR BREAKER exercise.

Problem Statement

The work accomplished last year on the AFIT Virtual Cockpit laid a solid foundation on which to build; however a lot of work remained to be done. To be useful as an attack aircraft simulator, the Virtual Cockpit needed the ability to deliver a weapons load on target. A weapons delivery capability required the weapons to be visible, move, and interact with other objects in the simulation. Conversely, weapons in the simulation had to be able to interact with the Virtual Cockpit. This required the Virtual Cockpit to evaluate the effects of weapons detonation on its systems. For the Virtual Cockpit to participate in future WAR BREAKER exercises, it must be DIS v2.0 compliant. Students from previous years made the design decision to ignore the curvature of the earth in all position and terrain calculations. DIS v2.0 no longer allows this assumption to be made, and thus greatly increased the complexity of many aspects of the simulation. Finally, the frame update rate of the version of the Virtual Cockpit was less than seven frames per second. The frame rate of the Virtual Cockpit must be increased to at least 15HZ. This is the minimum rate that will allow users to interact with the system without causing over correction.

Scope

During my research, I explored ways to add the ability to deliver weapons to the Virtual Cockpit, increase the simulation frame rate, and improve the appearance of the Virtual Cockpit. My primary focus was on the creation of a Weapon Controller. I intended the Weapon Controller to separate workings of the weapons from the rest of the Virtual Cockpit. This separation allowed me to try different approaches in creating the weapons while making only minimal changes to the existing code structure.

The addition of the weapons required modifying several parts of the existing version of the Virtual Cockpit. The first change was to the structure of the airplane object to integrate the Weapon Controller. While this required only adding the call to the Weapon Controller update function, the interaction of the network update function, the RADAR update, the HUD update, and the Weapon Controller update had to be analyzed to ensure these functions were working with the latest possible information about the simulation. I also had to change the Head-Up Display (HUD) to give proper targeting information to the pilot. Basic targeting information includes a Continuously Computed Impact Point (CCIP) for bomb delivery, a Target Designation Box (TDB), and a gun sight. When released, each bullet, bomb, or missile becomes an independent object in the overall simulation. This means each weapon has its own geometric model and methods for movement. Additionally the weapons must generate their own network messages.

Improving the simulation frame rate was a constant consideration in all Virtual Cockpit development. The primary method of improving the frame rate consisted of adapting the Virtual Cockpit to run within a simulation frame work called ObjectSim (30). ObjectSim relies extensively on the IRIS Performer environment. IRIS Performer is a software development environment that supports programmers implementing high performance, multi-processing graphics applications on Silicon Graphics products. It offers both high level facilities for visual simulation and virtual reality tasks and an application-neutral high-performance hardware-oriented graphics toolkit (28).

Improving the appearance of the Virtual Cockpit depended heavily on the success of adapting the simulation to Performer and ObjectSim. The previous version of the Virtual Cockpit had a significant problem with the frame rate (18:36). To help improve the frame rate of the first version, most of the polygons describing the F-15E model were removed, and the instrument panel became a simple texture mapped polygon. After adapting the Virtual Cockpit to run under ObjectSim the frame rate was high enough that more polygons were added to the Virtual Cockpit's description. Increasing the number of polygons in the model increased the realism but slowed down the frame rate.

Approach and Methodology

My first step in adding a weapons system to the Virtual Cockpit was to develop an understanding of the Virtual Cockpit v1.0. Any modification to the current system required a thorough understanding of the system and design decisions. The first changes to the system consisted of modifying the Virtual Cockpit to run under the ObjectSim framework. Next, I modified some of the instrumentation that already existed in the Cockpit. Primarily, I needed to upgrade the HUD to show the different weapons modes that are possible. While working with the Head-Up Display, I tried a different approach to rendering it. This approach created the HUD with Silicon Graphic's, Graphics Library line drawing function. Analysis of the Virtual Cockpit's frame rate showed this method was faster and improved the overall performance of the simulation.

I used an incremental approach for the addition of the Weapon Controller object. At each phase of development I would add the minimum functionality needed to test the next step. While I worked with the Weapon Controller, I decided to start with a small subset of all of the different possible weapons. The techniques I learned working with the subset of weapons I used later to add the full range of possible munitions.

The 20mm cannon was the first weapon I added. The cannon rounds moved along a simple ballistic path and checked each frame to see if they struck an object. I then added the Mk-82 bomb delivery capability since they also follow a ballistic path. The bombs were more complex than the cannon rounds for two reasons. First, the Head-Up Display had to produce a more complex display showing the probable point of impact. Second, I wanted the geometric models of external weapons like bombs and missiles to be visible in the simulation while they were still on the aircraft. Correctly placing the geometric models in the database hierarchy so that they could be seen by the user required significant research into the Performer database structure. The last weapon in the initial subset was the AIM-120 AMRAAM missile. Unlike the cannon rounds or bomb, missiles are powered during an initial boost phase immediately after launch. After the boost phase, the missile will still be able to maneuver but will have only a short period of time before it hits the ground. The boost phase and maneuvering added additional complexity to the model. The missiles also required interaction with the Virtual Cockpit's RADAR for target selection and tracking. After these weapons were functional, I applied the techniques I learned to add the remaining portion of the weapons.

Previous students designed this project to take full advantage of the Object Oriented Design Methodology. This design approach has several characteristics which models real world objects in software and helps the developers maintain the proper device interfaces. In addition, Object Oriented Design allows developers to add new objects one piece at a time. One important facet of this methodology is generalization, also known as inheritance. Generalization is the ability for a subclass object to inherit features of its superclass object and is sometimes known as an 'is-a' relationship (26:39). An example of this relationship would be, "An F-15 is a jet aircraft". The F-15 inherits features such as wings and jet engine from the superclass of jet aircraft, yet has special features not on all jet aircraft. Previous students chose the C++ computer language because it supports inheritance. Current versions of Ada do not support inheritance.

Hardware/Software

All of the equipment and software required to implement the Virtual Cockpit is available commercially. The equipment includes:

- Silicon Graphics four processor Onyx Workstation with Reality Engine graphics
- Polhemus "Looking Glass" Fiber-Optic Head Mounted Display
- Thrust Master WCS Stick, Throttle, and Rudder Pedals
- Polhemus Fastrack Position Reporting System

The software used for developing the Virtual Cockpit includes:

- C++: An object-oriented programming language
- MultiGen (Software Systems, Santa Clara, CA): A modeling tool for creating 3-D polygonal object descriptions

C++ is a high level programming language and is fairly platform independent. When newer and faster computers become available, transferring this system to the new computer will be straightforward.

Thesis Overview

The next chapter of my thesis provides additional background information. Chapters III, IV, and V discuss the design and implementation of my solution. Chapter VI contains the results of my research.

II. Background

Overview

When the Virtual Cockpit is broken down into its component technologies, it is a flight simulator, connected to a network, which uses virtual environment technology to present visual information to the user. Although none of these technologies is new, they have not been effectively combined in this type of application. This chapter presents background information concerning flight simulators, virtual environments, and distributed simulations.

Flight Simulators

Studies have shown that flight simulators are an efficient and effective means of training pilots. While the simulators can not replace flight time, if used in the proper manner, they can reduce the amount of time needed to master certain procedures. Analysis of 22 studies conducted on flight simulators between 1967 and 1977 showed that pilots trained in simulators required about half the flight time to master certain skills as compared to pilots trained only in aircraft. Part of the reason for the decrease in required flight time is that a simulator allows the student to practice a task more times in a given period than actual aircraft (23:195). With the operating costs of a simulator about eight percent of the actual aircraft, simulators can dramatically reduce training costs (25:235). However, these savings can be realized only if the simulator faithfully represents the performance of the actual aircraft.

Flight simulators have a long history and have been in existence almost as long as airplanes. Their purpose has traditionally been to train future pilots in an environment more benign than an actual airplane. Engineers have always employed the most advanced technology available in their construction, with varying degrees of success. The following paragraphs on flight simulators are summarized from Chapter 2 of *Flight Simulation* by J.M. Rolfe.

A faithful simulation of an aircraft in flight requires three elements (25:17):

1. A complete model, preferably expressed mathematically, of the response of the aircraft to all inputs from the pilot and from the environment.
2. A means of solving these equations in real-time.
3. A means of presenting the output of this solution to the pilot by means of mechanical, visual and aural responses.

Engineers have been trying to provide these three elements using various methods since 1910. These methods have been constantly evolving as technology increases. I have grouped the simulators into seven different generations depending upon the technology used to build them. These generations and their approximate time frames are:

1. Early simulators using actual aircraft (1903-1919)
2. Mechanical simulators with motion (1910-1930)
3. Fixed-based, mechanical analog (1928-1945)
4. Fixed-based, electrical analog (1941-1960)
5. Fixed-based, digital computers with no visual system (1950-1957)
6. Full Motion, digital computers with closed circuit television (1955-1978)
7. Full Motion, digital computers with computer generated images on projection screens (1974-present)

The first generation simulators were typically actual aircraft mounted on a universal joint. The wind blowing over the wings generated lift. The lift from the wings brought the body of the simulator upright on top of the universal joint. It was then up to the student to provide the control inputs to keep the simulator in equilibrium. During this generation the first requirement of a faithful simulation was poorly understood. Lack of a through understanding of aerodynamics accounts

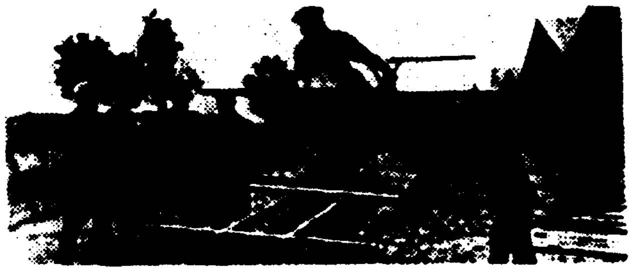

Figure 1. Antoinette trainer
(25:16)

for the many different aircraft designs, and the number of designs that failed. The second and third requirements were provided by an actual aircraft and were therefore very realistic. This type of simulator was effective, but it did have its drawbacks. These simulators were large, the size of the actual aircraft. They required a steady source of wind to generate lift. Finally, because they were mounted on the ground, they could not simulate steep dives or banks. (25:15)

The next two generations of simulators tried using various mechanical means of providing the first two required elements of an effective flight simulator. The mechanical simulators, such as the Antoinette trainer shown in Figure 1, were also mounted on universal joints. The earliest versions had the instructor pilot providing the motion cues to the simulator and the student trying to correct the motion using controls connected to wires and pulleys in the base. The training value for these devices was low because the motion of the simulator was not based on any true aircraft model, and the student received immediate response from control inputs. (25:16)

By 1920 aeronautical engineers could satisfy the first requirement of a faithful flight simulation and produce a mathematical model of an aircraft in flight. However, no known method could solve the equations of motion in real time. Instead, the engineers created mechanical analog simulators that no longer tried to imitate the motion of aircraft, but concentrated on instrumentation flight. These simulators

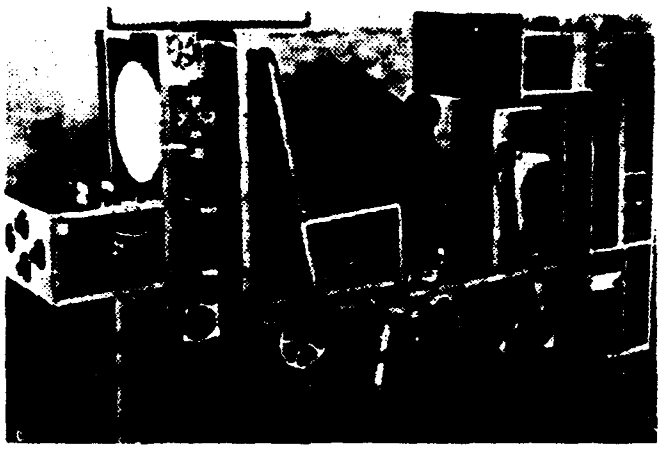
Figure 2. Electronic Analog Simulator
(25:30)

employed mechanical and pneumatic techniques to change the instrument readings based upon the pilot's input. The instrument responses were based on an empirical model instead of a mathematical model of the specific aircraft. Engineers adjusted the simulator's cams, gears, and pneumatic pumps until the instrument readouts were similar to an actual aircraft in flight. (25:19-27)

The advancement of electronics during World War II brought about the first electronic, analog computers that could satisfy both the first and second requirement of a faithful flight simulation. The Curtiss-Wright Z-1, shown in Figure 2, was an example of this type of simulator. These analog computers could solve the basic equations of motion for an aircraft in real time. Instead of doing numeric calculations, the electronic analog computer performs a direct simulation of the physical device. Each section of the computer models a particular portion of the simulated aircraft. Unlike a digital computer which manipulates the discrete states of electronic switches, the analog computer changes the value of a continuous signal, usually electrical voltages that vary with time (17:2). The voltages produced by the analog computer reflect information about the state of that portion of the aircraft being modeled. For example, the analog computer representing the wings produces a signal that would be processed to determine the amount of lift being generated. If the lift is greater than the aircraft's weight then the signal going to the altimeter gauge would increase, reflecting a gain of altitude.

Due to the size and weight of the vacuum tubes and wires, these simulators did not provide any motion cues. However, for the first time the values shown on the instruments were based on the aerodynamics of the simulated aircraft. The electronic nature of these simulators also enabled the instructor to modify the simulated conditions easily and precisely, allowing the training crew to practice emergency procedures. The success of the electronic analog simulator led to increased requirements for greater accuracy. The accuracy of the electronic analog simulators was increased by adding more analog circuitry, but this approach soon reached the point of diminishing returns. The small cumulative errors inherently present in the analog circuits multiplied as the amount of circuitry increased. Such errors could be introduced by an amplifier boosting a signal too much or a variation in the conductivity of the copper wires. These errors canceled the gains in accuracy. The additional number of components also decreased the mean time between failure rate for these simulators. (25:32)

The introduction of digital electronics solved the accuracy problem presented by analog circuitry. Instead of the analog computer's continuous signals, the digital computer used discrete signals. These discrete signals can not be skewed by small variations in the hardware. There were several digital flight simulation projects started in the early 1950's. These projects, like the Navy's Universal Digital Operational Flight Trainer, used computers specially designed to solve aircraft equations of motion. However, as general purpose computers became faster and more capable, the use of the specially designed computers declined. The digital computer is still the basis of a modern flight simulator and as digital computer technology advances, flight simulators will also advance. (25:33)

After the digital computer fulfilled the first two requirements of using a mathematical model of aircraft characteristics to solve the equations of motion in real time,

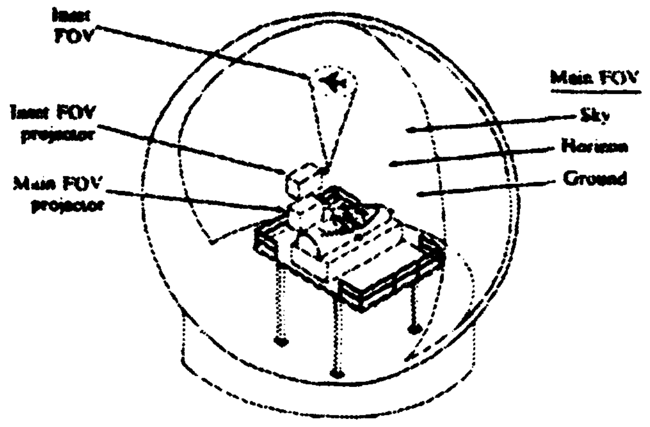

Figure 3. Flight Simulator with Computer Generated Image on a Dome (25:150)

the third requirement of presenting output to the user received more attention. One advancement was the reinvention of full-motion simulators, which create accelerations in six degrees of freedom. The presentation of realistic images has proven to be more difficult. For over 25 years closed circuit television cameras moving over a scale model terrain board proved to be one of the most successful methods. However, the terrain board required substantial logistical support. First, the terrain model boards were typically 40 feet long and 15 feet high. The large size of the boards dictated a large building and made training over different terrains impractical due to the necessity of substituting boards. The television cameras required servo motors and a large number of bright lights. The heat generated by the lights drove up the cost of air conditioning the large building. (25:131)

By 1980 there were over 300 flight simulators that used computer generated images projected onto spherical screens for the visual system. These systems use conventional 3D computer graphic principles to generate the out-the-window view (Figure 3). The Main Field of View (FOV) projector displays a low detail image of the sky, horizon, and ground onto the screen. The low detail image provides the pilot with altitude and orientation cues. The Inset FOV projector displays a smaller, more detailed image of objects of interest, such as other aircraft. The position of the simulated aircraft and other objects determine the image projected on the screen. The detail of the image is directly linked to the computational power of the computer generating the image (25:131). The development of virtual reality technology may provide a replacement for the large projection screens, thus making the simulators less expensive and smaller.

Virtual Environments

In a paper presented in 1965 to the International Federation of Information Processing Societies, Ivan Sutherland proposed the concept of the Ultimate Display. Dr. Sutherland's idea was that a computer display should interact with as many of the user's senses as possible. Currently a computer passes most information to a user through the sense of sight. The user sees what is happening on the computer screen. High quality digital audio technology brings the user's sense of hearing into play, but it is used much less frequently. Dr. Sutherland proposed that other senses could be included, and a person using the Ultimate Display would not be able to distinguish the computer generated environment from the physical world (32:508).

Computer researchers have used this description of the Ultimate Display as the inspiration for the development of virtual environments. Virtual environments have potential uses in many different areas. People from many diverse fields could use virtual environment simulations to sharpen their skills. Surgeons could practice before performing difficult operations or fighter pilots could practice missions before flying. Scientific visualization and the entertainment industry also provide incentives for the development of virtual environments (1:168). Twenty-eight years after the initial concept, the technology to implement the Ultimate Display still does not exist. However, impressive advances in the areas of computer workstations, head-mounted displays, and tracking equipment are bringing virtual environments closer to reality.

The computer is the engine on which all virtual environment development depends. The cost of special workstations with optimized hardware for drawing images is decreasing at a rate of 50% per year (1:168). As the performance to price ratio increases, the newer, less expensive computers create more realistic images. However, one drawback to these new computers for virtual environment applications concerns latency. Latency is the delay between where the user is currently looking and the corresponding change in the display created by the computer. The architecture of workstations used to generate graphic images are 'pipelined'. This pipeline is similar to a car assembly line, except a computer image is being built, and is a cause of some latency. Different portions of the image description have different calculations being performed on them at the same time. Normally this reduces the amount of time it takes to draw the image. However, when the image description changes rapidly all of the old image must be cleaned out of the pipeline before the new image description is processed. The delay needed to clean out the pipeline is a significant source of latency, and researchers are investigating various ways of reducing it (1:168).

Head-mounted displays are another area of much research in virtual environments. The concept behind a head-mounted display is to present a computer generated, three-dimensional image to the user in such a way that the user feels immersed in the image. Researchers are currently focusing on the creation of light weight, maneuverable head sets that put the image directly in front of the user's eyes. The current generation of head-mounted displays primarily use color liquid crystal displays, which typically have a resolution of 200 pixels high and 300 pixel across (1:169). A different approach being researched at the Air Force Human Resources Laboratory uses four light-valve projectors to transmit each eye's image through fiber optic cables to the head-mounted display. The fiber optic cables project the images on to small mirrors in front of the user's eyes. The field of view on this display is 135 degrees and a resolution of 2-3 arc minutes for each pixel (13:262). The Air Force Institute of Technology's graphics lab currently uses a head-mounted display based on a similar approach. The Advanced Research Project Agency is sponsoring the development of higher resolution displays. These displays will use liquid crystal display systems with a monochrome resolution of 1280 pixels by 1024 pixels. The color resolution of these displays will be less than the monochrome resolution. The development of the new displays should be completed in the two years (1:169).

In addition to projecting the computer generated image for the user to see, the computer creating the virtual environment must know where the user is looking. This type of input to the computer requires specialized hardware to track the movement of the user's head. There are several systems commercially available; the most common systems use magnetic fields emitted from a small antenna in a fixed position. A device placed on a head-mounted display detects the magnetic fields. Any movement by the user causes a change in the field detected by the receiving antenna (1:167). The tracker interprets the changes in field strength as movement and reports the new position and orientation of the user's head to the computer.

These magnetic systems need improvement. Due to the very nature of the magnetic fields, these trackers are sensitive to metal objects in the area. The metal objects distort the shape of the magnetic fields and cause errors in the reported position (6:46). The magnetic trackers also have a very limited effective range, usually less than one meter (1:167).

Current digital audio technology is ready for large scale integration into virtual environments. The challenge is to incorporate sound in the virtual environment so that it is believable. The two main categories of devices for computer generated sound consist of samplers and synthesizers. Samplers can replay almost any pre-recorded sound. However, samplers require a large amount of storage media for the pre-recorded sound and have problems reproducing sounds in real time. Synthesizers rely on analog or digital sound generation techniques. Synthesizers are more limited in the sounds they can produce, but are more adaptable to real-time manipulations.

Once the sound description is created, the computer must process the description so that the sound fits into the virtual environment. The sound emitted by an approaching object in the virtual environment should have the appropriate Doppler shift, or the sound coming from an object behind the user should sound as if it is behind him or her. Students at the Air Force Institute of Technology are conducting research in the area of audio cue generation. Current work includes the use of a Polhemus IsoTracker to monitor the user's head position and an audio localization cue synthesizer. The synthesizer uses information about the user's position to modify the audio cue to give it the proper 3-D effect (27:459). This type of processing is very intensive and requires approximately 300 million instructions per second to generate the proper sounds. Research needs to be conducted on models of perception to make sure useful and proper acoustical cues are being generated (1:170).

Fidelity versus Rendering

One of the fundamental objectives of virtual environments is to create an illusion of immersion in the environment for the user. For this sense of immersion to work, a view of the scene reflecting the position and orientation of the user's head must be presented to the user. The head-mounted display and magnetic tracking system previously described fulfill these requirements. However, the speed at which the computer can read the tracking device and render the image for the display is critical for the illusicn. Steve Bryson demonstrates the relationship between image update rate and the quality of the illusion (Table 1).

Update Rate in Frames/Second	Quality of Illusion
Less than 5	Fails completely. (Looks like a succession of still photos.)
Between 5 and 10	Works poorly.
Between 10 and 15	Works well.
Between 15 and 25	Works convincingly.

Table 1. Quality of Illusion
(5:1.3.3)

For a simulation to work effectively, it requires a minimum amount of fidelity. This fidelity represents how close an object's actions and appearance correspond to the same object in the real world. Increasing the accuracy of an object's appearance requires adding more polygons to the description and possibly texture mapping the surface. Accurately portraying how an object moves requires detailed equations of motion and numerous calculations. Adding polygons and texture maps, and solving detailed equations to enhance fidelity requires more CPU time and slows down the simulation. Obviously anyone who tries to develop a graphic simulation must make some tradeoffs between the fidelity of the simulation and the rendering speed of the graphics.

Distributed Interactive Simulation

When we had gone so far across the ice
that it pleased my Guide to show me the foul creature
which once had worn the grace of Paradise,
he made me stop, and, stepping aside, he said:

Now see the face of DIS! This is the
place where you must arm your soul against all dread.

Do not ask, Reader, how my blood ran cold
and my voice choked up with fear. I cannot write it:
this is a terror that cannot be told. - Dante (3:283)

For a simulation to be truly useful and interesting, a person must not only interact with objects in the virtual environment but also with other people in the same environment. This interaction requires different simulators to communicate with each other via a computer network. The Advanced Research Project Agency is sponsoring the development of the Distributed Interactive Simulation (DIS) protocol to facilitate interaction between simulators.

DIS is a time and space coherent synthetic representation of world environments designed for linking the interactive, free play activities of people in operational exercises. The synthetic environment is created through real-time exchange of

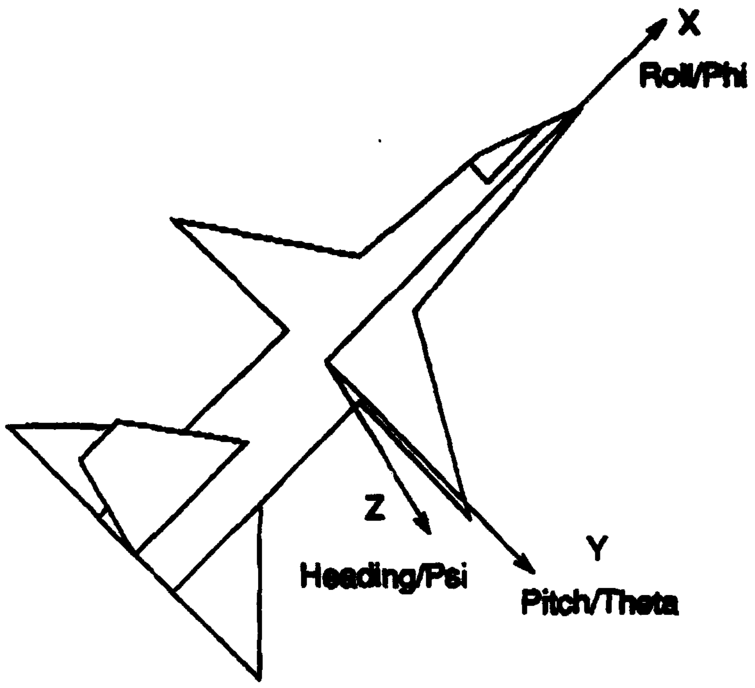

Figure 4. DIS Entity Coordinate System
(11)

data units between distributed, computationally autonomous simulation applications in the form of simulations, simulators, and instrumented equipment interconnected through standard computer communicative services. The computational simulation entities may be present in one location or may be distributed geographically (14:1).

The DIS protocol defines the simulation environment, communications, and fidelity requirements. Coordinate system definitions are a fundamental part of the simulation environment. DIS version 2.0 defines two separate coordinate systems, one for all the entities in the simulation, and one for the simulated world. The entity coordinate system gives a standard orientation for all the geometric models in the simulation. The entity coordinate system, shown in Figure 4, uses a right-handed Cartesian coordinate system whose origin is the center of the entity's bounding volume. The positive X axis goes out the front of entity, the positive Y axis out the right side, and the positive Z axis points down. The basic unit of measurement

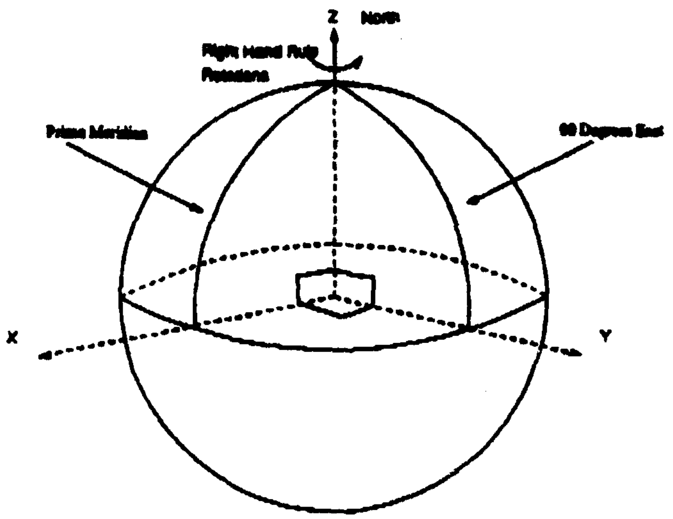

Figure 5. Geocentric Cartesian Coordinate System

is the meter (14:5). Engineers commonly use this coordinate system when working with aircraft aerodynamic forces (25:50).

The World Coordinate System is also a right-handed, geocentric, Cartesian coordinate system, shown in Figure 5. Its origin is the center of the earth with the positive X axis passing through the equator at 0 degrees longitude and the positive Z axis passing through the North Pole. The basic unit of measurement is the meter, and the World Geodetic System 1984 defines the size and shape of the earth (14:5). This coordinate system has several advantages such as the inclusion of satellites (16) and allowing simulations to take place anywhere in the world. It does present several challenges also. Among the challenges are adapting aircraft equations of motion to compensate for the curvature of the earth and determining which direction is down in the simulation.

In a distributed simulation, each user is hosted on his or her own computer. In addition to hosting a user, each computer must also keep track of the location of the other users. The computer uses a prediction model called dead reckoning to prevent the network from being overloaded (4:157). Dead reckoning uses an approximation of a user's position and orientation to display the information in the simulation.

A host machine keeps the dead reckoned position of everyone in the simulation in addition to the actual position of the local user. When the local user's dead reckoned position differs too greatly from his actual position or a specified amount of time has elapsed since the last update, the host computer transmits an updated position over the network.

The basic means of communication in a distributed interactive simulation is a Protocol Data Unit or PDU. In DIS version 2.0 there are 27 separate types of PDUs (14:22). The three most important PDU's are the Entity State PDU, Fire PDU, and Detonation PDU and are the only ones the Virtual Cockpit broadcasts. The Entity State PDU contains information about an object's identification, position, and operating condition. The Fire PDU is issued when a simulator releases a weapon of any type. The basic information in the Fire PDU is the issuing simulator identification, the weapon type, where it was released, and the target identification. A Detonation PDU always follows a Fire PDU. The Detonation PDU contains information on where the weapon detonated and whether or not the weapon hit its target.

Each local simulator on a DIS network determines if a weapon launched by its user hits anything in the simulation since the decision on when and where to release a weapon was made by the local simulator's user based on information the local simulator had at the time (14:28). This fact may seem unusual since an object may not have the same position on the local simulator as it has on the other simulators in the network. The difference in an object's position is caused by errors in dead reckoning and the time lag caused by sending, receiving, and interpreting Entity State PDUs. The damage inflicted on the target is left for the target's simulator to evaluate, and it is made known by broadcasting its condition (undamaged, slightly damaged, etc.) in an Entity State PDU.

III. System Design

Overview

The next three chapters discuss how I met the goals defined in the Scope section of Chapter I. This chapter provides some additional background information on Iris Performer and ObjectSim and how they influenced the design of the Virtual Cockpit.

Performer

The Virtual Cockpit creates series of computer generated images shown to a user in rapid succession. The images need to be drawn fast enough that the user interprets them as smooth motion. Therefore the Virtual Cock‐‘‘s most basic, fundamental task is lighting individual pixels as fast as possible. Hopefully, the user interprets the pattern formed by the pixels as a flight simulation. A pixel's color is dependent on a large number of mathematical equations that are abstractly represented as polygons. My task was to work with the algorithms that move and orient the polygons. The students in the graphics sequence chose the Iris Performer software to convert the polygon position information into pixel shading information for several different applications. Iris Performer is a software development environment consisting of several function libraries written in C. It supports the implementation of high performance, multi-processing graphics applications on Silicon Graphics products. The Performer Libraries offer high-level facilities, such as intersection tests and view parameter definition, for visual simulation and virtual reality tasks. (19)

One of the high-level Performer facilities organizes the polygon descriptions into a hierarchi‐al database. This database is contained in a standard tree structure (Figure 6). The polygon descriptions are the leaf nodes at the bottom of the database and can be created with a variety of visual database modelers such as WaveFront and MultiGen. The database's parent nodes are data structures that contain 4x4

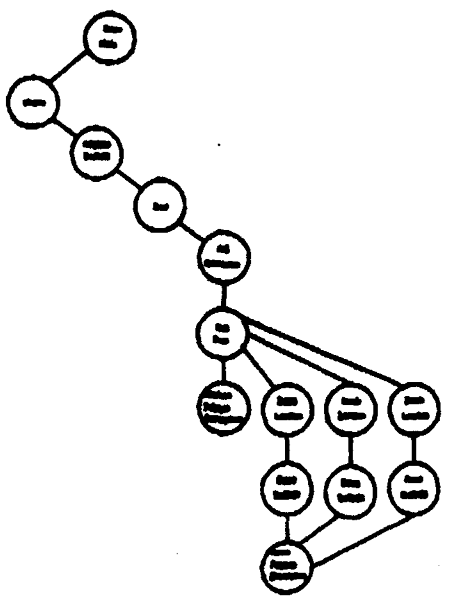
Figure 6. Nodes in the IRIS Performer Database

matrices used for transformations. To draw a scene, Performer traverses the tree-structured database and applies the matrix transformations in a parent node to its children nodes. By the time Performer traverses the database down to a polygon description, it has processed all of the necessary matrix transformations to place the polygon in the scene. The faster the computer can process the matrix transformations, determine which polygons are visible to the user, and draw them on the screen, the smoother the motion in the simulation. One way of improving the drawing speed is to use more than one processor.

A graphics application using the Performer Libraries can create up to three processes which are used to create a visual simulation. The processes are called the Application process, the Cull process, and the Draw process. The Application process is initialized first and contains the main loop for the simulation. The Application process in the Virtual Cockpit is responsible for several tasks including:

- Getting user inputs from the Polhemus Fastrack.
- Moving the polygon description of the aircraft based on user inputs.

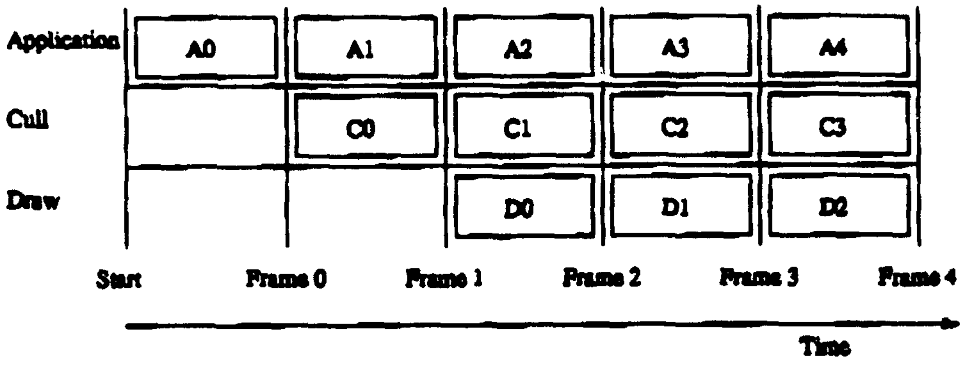

Figure 7. Performer Application/Cull/Draw Pipeline
(19:7-15)

- Moving the polygon description of the released weapons.
- Determining the effect of weapon detonations on local objects.
- Determining if objects are illuminated by the RADAR.
- Calculating the cues for the HUD and RADAR.

The Cull process traverses the Performer database and determines which polygons are visible in the user's current view. It puts the polygons that are visible into a data structure called the draw list. The Draw process then uses the polygons in the draw list to determine the color of the individual pixels.

The Application, Cull, and Draw processes form an assembly line, with each process working on information for a different frame (Figure 7). The Application process gets the user inputs and changes the transformation matrices in the Performer Database. After the Application process finishes, the Cull process creates the draw list from a copy of the Performer Database. While the Cull process is creating the draw list, the Application process starts modifying the Performer Database for the next frame. After the Cull finishes the draw list, the Draw process begins to draw the scene. While the Draw process is working on Frame N, the Cull process is working on a draw list for Frame N+1, and the Application process is working on Frame N+2. This pipeline minimizes the time the Draw process must wait for a draw list.

The speed of the Draw process is usually the limiting factor of the frame update rate. After rendering the polygons, the Draw process must also render the HUD and RADAR displays. The information required to render the HUD and the RADAR is not contained in the Performer database; instead they are rendered using SGI Graphics Library function calls. The HUD and the RADAR must be rendered by the Draw process because the design of the SGI hardware requires a graphics pipeline to be controlled by only one processor at a time. To reduce the number of calculations the Draw process must execute, as many of the calculations as possible needed to draw the HUD and RADAR cues are done in the Application process. This information is then passed to the Draw process through the use of shared memory and locks.

Shared memory is simply memory that can be accessed by multiple processors. The processors in a Silicon Graphics Onyx workstation operate independently of each other, and could cause conflicts by attempting to access the same shared memory simultaneously. One example of the possible conflict is the Application process writing the values of the HUD cues to the shared memory while the Draw process is trying to read them. The Application process may have written out the first two bytes of a float variable when the Draw process tries to read it. The value read by the Draw process would not be correct. Therefore data structures in shared memory must be protected against being accessed by more than one processor at a time. The required protection is available through the SGI operating system support in the form of locks.

Locks are a special type of variable that processes use to protect critical sections of code. Locks are declared in shared memory and cannot be accessed by more than one process at a time. When a process reaches a critical section of code, typically reading or writing to shared memory, it tries to set a specific lock. If the process is successful in setting the lock, it goes ahead and executes the critical section of code. If the process is not successful because some other process has previously set the

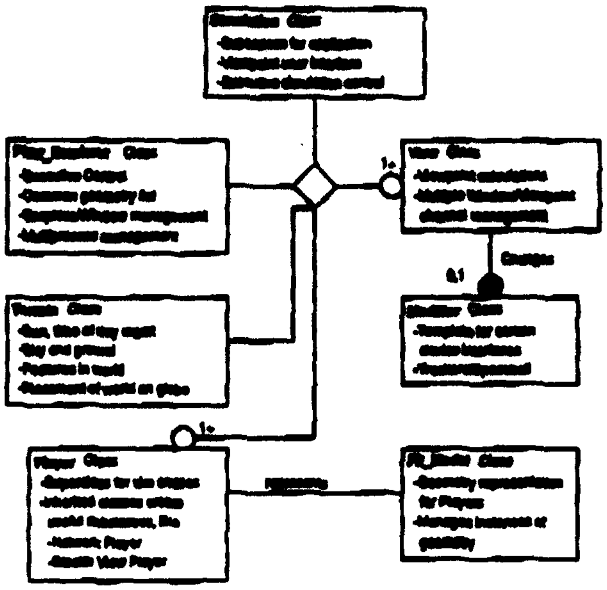

Figure 8. ObjectSim Class Relations
(30)

lock, the unsuccessful process must wait until the controlling process resets the lock. In my experience, only the code to actually copy into or out of a shared memory structure should be inside the lock. This design minimizes waiting time for other processes and ensures the unlock function will be executed when the shared memory is no longer in use.

ObjectSim

ObjectSim is a library of C++ classes that encapsulate much of the Performer functionality in a more abstract form. At its highest level, ObjectSim is a generic visual simulation that a developer can use as a basis for a specific simulation. The ObjectSim classes shown in Figure 8 represent the objects typically found in any visual simulation. The Player Class is the super-class of the entities that interact in a simulation. Their basic characteristics consist of a position (X,Y,Z) and an orientation (heading, pitch, roll). The Airplane object and all the weapons in the Virtual Cockpit simulation are Player objects. Each Player object also has a pointer to an object of the Flt_Model class. The Flt_Model class manages the polygon models for a Player object. It also allows several Player objects to share one instance of a polygon model. The model is placed in different locations in the simulation by having different transformations in the pfDCS nodes before getting down to the geometry node. One example of a single geometry description is shown in Figure 6. Each bomb has its own location on the plane (Bomb Location) and a Dynamic Coordinate System (Bomb RotDCS) for moving independently in the scene. However, they all share the same polygon description for a bomb.

The View class controls what the user sees in the simulation and contains several attributes useful for defining a view into the simulation. These attributes include a position (X,Y,Z), a direction (heading, pitch, roll), and a horizontal and vertical field of view (degrees). Each View class object may have a Modifier class object that can change the position or direction of a view by acting as the interface to some type of input device. Typical Modifier class input devices include the Polhemus Fastrack Magnetic Tracker, Dimension 6 spaceball, and the mouse.

The Terrain Class manages many environmental aspects of a simulation. The environmental aspects include the polygon description of the earth where the simulation is being held, the placement of cultural features such as bridges and buildings, and the ambient light levels based on time of day. The Terrain Class is also responsible for converting positions and orientations based on a local coordinate system into positions and orientations based on the WGS-84 coordinate system (11).

The Pfmr_Renderer Class contains the primary iterative loop of the simulation. This loop gets input from the user, tells the objects in the simulation to move, and then outputs the appropriate graphics to the display. Finally, the Simulation Class puts all of the pieces needed for a simulation together by declaring instances of the appropriate classes.

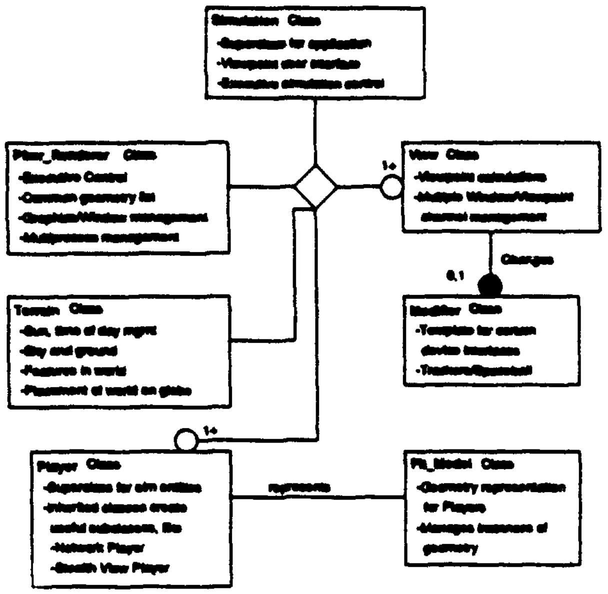
Simulation Class
-Superclass for application
-Viewpoint user interface
-Executive simulation control
Pfmr_Renderer Class
-Executive Control
-Common geometry list
-Graphics/Window management
-Multiprocess management
1+
View Class
-Viewpoint calculations
-Multiple Window/Viewpoint channel management
Changes
0,1
Modifier Class
-Template for custom device interfaces
-Trackers/Spaceball
Terrain Class
-Sun, time of day night
-Sky and ground
-Features in world
-Placement of world on globe
1+
Player Class
-Superclass for sim entities
-Inherited classes create useful subclasses, like
-Network Player
-Stealth View Player
represents
Flt_Model Class
-Geometry representation for Players
-Manages instances of geometry

Figure 8. ObjectSim Class Relations
(30)

ObjectSim is a library of C++ classes that encapsulate much of the Performer functionality in a more abstract form. At its highest level, ObjectSim is a generic visual simulation that a developer can use as a basis for a specific simulation. The ObjectSim classes shown in Figure 8 represent the objects typically found in any visual simulation. The Player Class is the super-class of the entities that interact in a simulation. Their basic characteristics consist of a position (X,Y,Z) and an orientation (heading, pitch, roll). The Airplane object and all the weapons in the Virtual Cockpit simulation are Player objects. Each Player object also has a pointer to an object of the Flt_Model class. The Flt_Model class manages the polygon models for a Player object. It also allows several Player objects to share one instance of a polygon model. The model is placed in different locations in the simulation by having different transformations in the intermediate nodes before getting down to the geometry node. One example of a single geometry description is shown in Figure 6. Each 'bomb has its own location on the plane (Bomb Location) and a Dynamic Coordinate System (Bomb RotDCS) for moving independently in the scene. However, they all share the same polygon description for a bomb.

The View class controls what the user sees in the simulation and contains several attributes useful for defining a view into the simulation. These attributes include a position (X,Y,Z), a direction (heading, pitch, roll), and a horizontal and vertical field of view (degrees). Each View class object may have a Modifier class object that can change the position or direction of a view by acting as the interface to some type of input device. Typical Modifier class input devices include the Polhemus Fastrack Magnetic Tracker, Dimension 6 spaceball, and the mouse.

The Terrain Class manages many environmental aspects of a simulation. The environmental aspects include the polygon description of the earth where the simulation is being held, the placement of cultural features such as bridges and buildings, and the ambient light levels based on time of day. The Terrain Class is also responsible for converting positions and orientations based on a local coordinate system into positions and orientations based on the WGS-84 coordinate system (11).

The Pfmr_Renderer Class contains the primary iterative loop of the simulation. This loop gets input from the user, tells the objects in the simulation to move, and then outputs the appropriate graphics to the display. Finally, the Simulation Class puts all of the pieces needed for a simulation together by declaring instances of the appropriate classes.

Utilizing Other Processes

In addition to the three processes created by Performer, the Virtual Cockpit generates two other processes, the Network process and the HOTAS Read process. The Network process receives information on the network and updates the position of other objects in the simulation. The information coming in from the network contains the positions of objects controlled by other simulators, and weapon fire and detonation information (29). The last process created by the Virtual Cockpit reads information coming from the Hands-On Throttle and Stick (HOTAS).

Originally the Application process was responsible for reading the RS-232 serial port connecting the HOTAS to the SGI workstation. In the initial version of the Virtual Cockpit, this worked well because tests indicated that the simulation was graphics bound. This means that the limiting factor on the frame update rate was drawing the polygons in the scene. Tests conducted after converting the Virtual Cockpit to ObjectSim, which greatly increased the frame rate, and adding both the RADAR and weapon systems, which slowed down the Application process, indicated that the simulation was not running as fast as was possible. The CPU running the Application process spent over 20% of its time waiting for input from the HOTAS. This idle time caused the total time required by the Application process and the Cull process to exceed the time required by the Draw process. In effect, the Draw process had to wait for its next draw list.

Reading the HOTAS in the Application process was taking too long, so I decided to move the HOTAS read function to another process. The ObjectSim Pfmr_Renderer Class has several function call backs in its design. These call backs are simply calls to specific functions in the Simulation Class. These functions do not have a predefined purpose and are available for customization of the simulation. A person creating a simulation can perform some task at a specific time in the simulation, such as after the draw list is made, by inserting the code into the appropriate function. The Pfmr_Renderer Class will then call the function at the correct time.

One of these call backs is on the Cull process, and is called after the process creates the visible polygon list for the Draw process. This process ran the fastest, and the CPU loading for this process was less than either the Application or Draw processes. Because of the faster execution time, I thought that the Cull process call back would be a good location for the HOTAS read. I restructured the HOTAS object to work with shared memory, and got the HOTAS read function working on the Cull process. After running some tests I discovered that reading the HOTAS slowed down the Cull process too much. Because it was now creating the draw list

	Average	Standard Dev.	Variance	Minimum	Maximum
Read in App with 100 objects	7.3	6.1	37.3	0.6	30.1
Read in Separate process with 100 objects	2.6	2.5	6.2	0.6	13.8
Read in App with 0 objects	4.8	4.0	16.1	0.6	20.3
Read in Separate process with 0 objects	1.9	1.1	1.4	1.1	6.3

Table 2. Application Process Execution Times (in ms)

and reading the HOTAS, the cull process could not finish creating a list of visible polygons before the Draw process needed it. Since the Draw process did not have any polygons to draw, the entire simulation disappeared. The Cull process is under very demanding real-time constraints and should not be given any extra tasks.

The best solution was to create the Read process in addition to the Application, Cull, and Draw processes. The Read process is a simple five line program that contains an infinite loop. The code inside the loop sets a shared memory lock used for synchronization and calls the HOTAS read function. The HOTAS read function gets the data from the RS-232 serial port and then sets a second lock to prevent simultaneous access of a shared memory structure. The function then copies the data into the shared memory structure and resets the simultaneous access lock. When the Application process is ready for the HOTAS inputs, it sets the simultaneous access lock, copies the values in the shared memory structure out to local variables and then resets both shared memory locks. Shared memory locks are not typically set by one process and reset by another, but it was necessary in this case to synchronize the two separate processes.

The synchronization lock allows the the Application process to run as fast as possible. In order to ensure the fastest possible execution, the Application process must be able to access the shared data structure on demand. When both processes are running without the synchronization lock, the Application process may still have to wait for the Read process to finish writing the data to the shared data structure. The synchronization lock allows the Read process to update the shared data structure once. The Read process must then wait for the Application process to reset the synchronization lock before calling the HOTAS read function again. While the Read process is waiting, the Application process has unimpeded access to the shared data structure.

The Application process execution times for both before and after the creation of the HOTAS Read process are summarized in Table 2. The data was taken from the pfDrawChanStats function over four different runs, each run lasting approximately two minutes. For the first two runs there were 100 objects being generated by another simulator on the network. The Application process required extra time on the first two runs to process data for the RADAR and to get the weapons targeting information. The data gathered indicates that creating the Read process cut the Application process execution time by 50%. Although the data being read by the Application process is not the most current HOTAS input, there is no noticeable lag because the frame update rate is typically 15 to 20 frames per second.

Improving Appearance and Adding Detail

Adapting the Virtual Cockpit to run in the Performer environment under ObjectSim increased its frame rate from approximately 7 frames per second to 30 frames per second. The higher frame rate allowed us to increase the detail of the simulation by adding more polygons. The first step was to substitute a much more detailed terrain model and a better model of the F-15E that surrounds the user. These changes were made by simply changing the data files read into the Performer database by the Virtual Cockpit. It was also decided that the instrument panel of the Virtual Cockpit should be redone and be modeled after an actual F-15E.

The Virtual Cockpit's original instrument panel (Figure 9) was a simple display consisting of six dials showing information such as airspeed and altitude. The actual

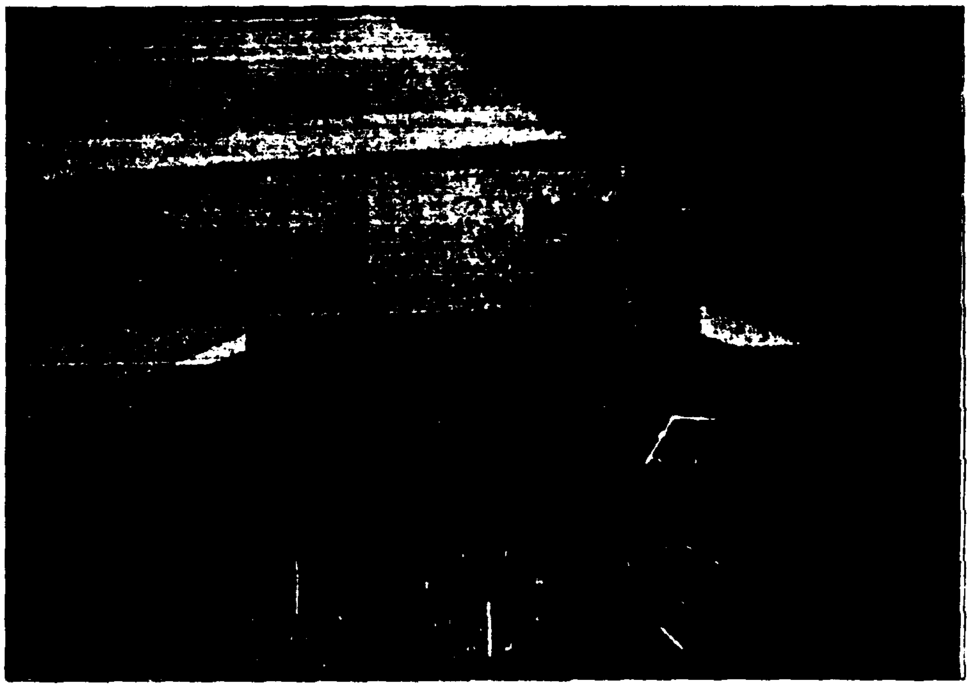

Figure 9. Virtual Cockpit v1.0 Layout

gauges were a texture map on a single polygon, and the needles were small polygons that overlaid the texture map. The first attempt at improving the Virtual Cockpit's instrumentation also used a texture map. The texture map was created by scanning in photographs of a real F-15E interior taken by Capt Alain Jones and Capt Matt Erichsen. After the image was scanned in, it was touched up using the ADOBE Photoshop software on a MacIntosh. I then used the final image as a texture map and placed it on the instrument panel. The realism of the image used contrasted poorly with the animated-quality of the rest of the simulation, and the two-dimensional projection of the texture map would also cause false depth cues if the Virtual Cockpit ever generated stereo images.

As an alternative to the texture map approach, I decided to model the instrument gauges and displays with polygons. In the first attempt using this method, I added the instrument polygon descriptions directly to the file containing the F-15E description. This approach also had its problems. The actual dials and gauges in

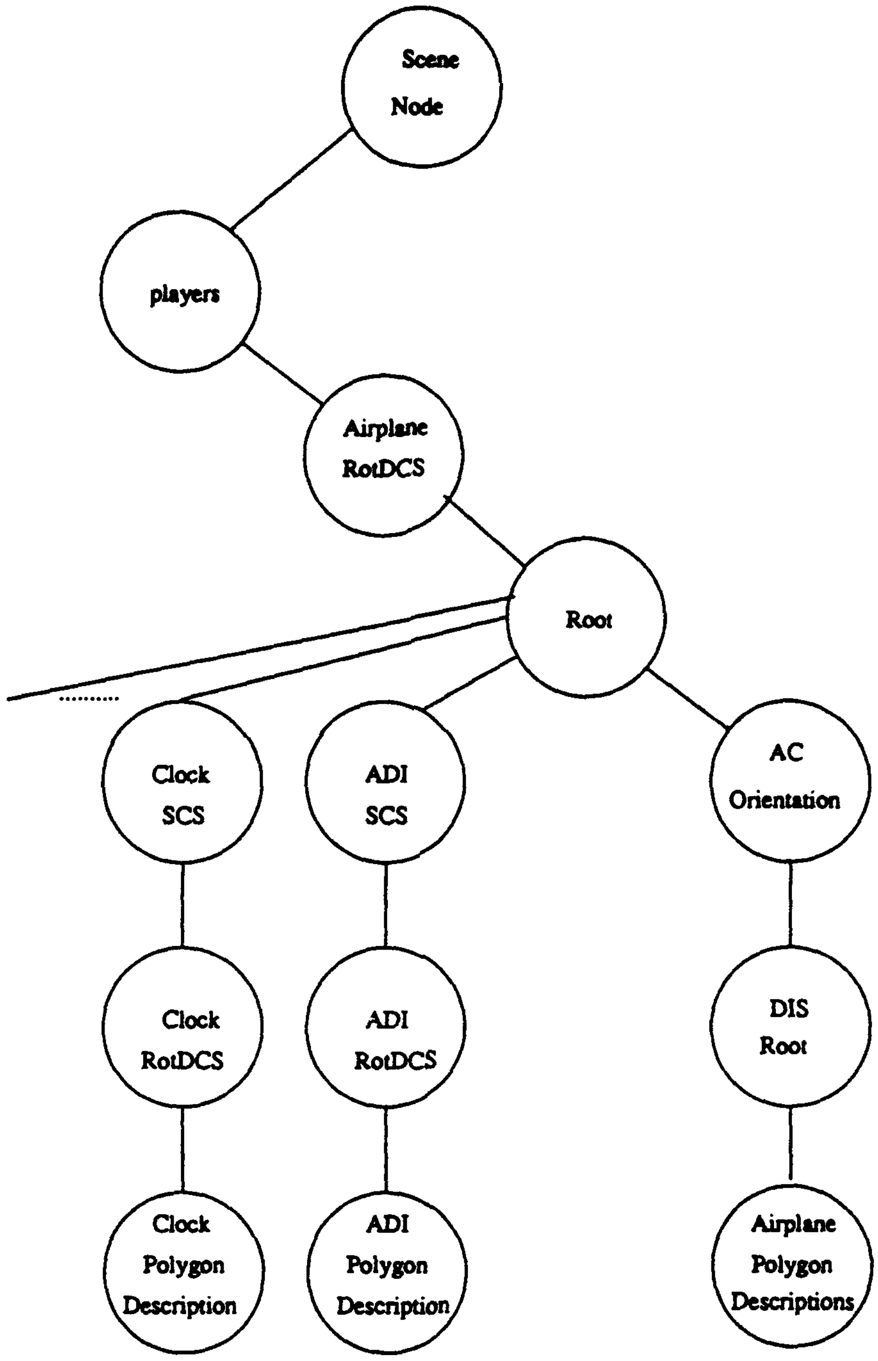

Figure 10. Instrument Descriptions in Performer Tree

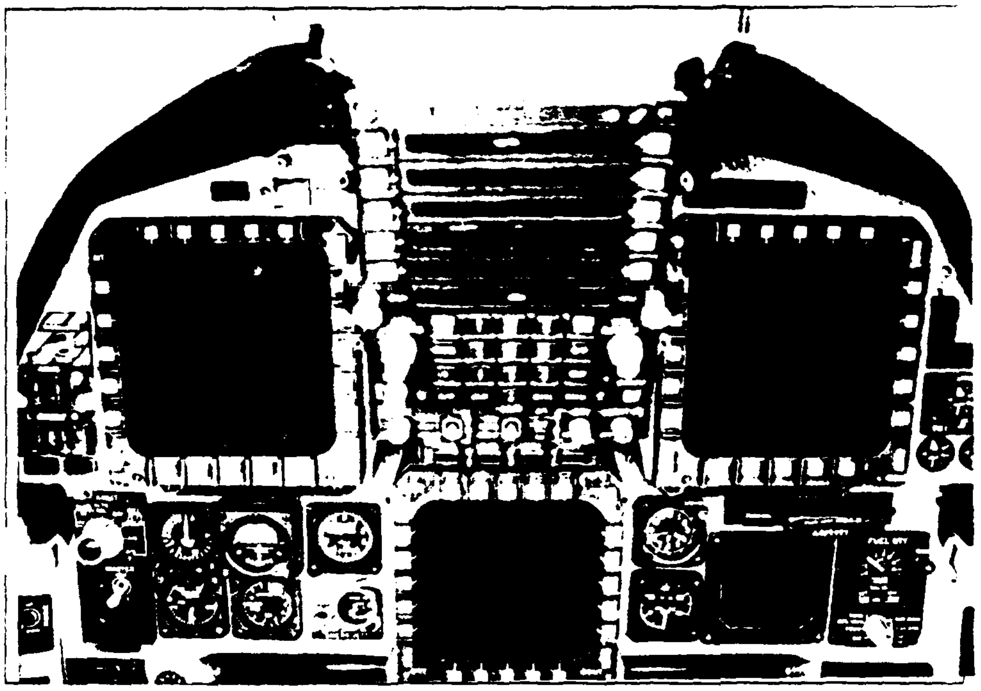

Figure 11. Actual F-15E Cockpit Layout

an F-15E are approximately five centimeters across, with the numbers inside only a few millimeters high. The polygonial descriptions of all the objects in the simulation are created using the MultiGen modeling software. Since version 12 of MultiGen does not have sufficient resolution to model polygons smaller than a millimeter, the numbers inside the gauges could not be modeled to scale.

The final solution was to use MultiGen and create each instrument as a separate description. An individual gauge was modeled as being one meter across to give the numbers inside the desired resolution. At start up time, the Virtual Cockpit reads all of the instruments' polygon descriptions and makes them children of the F-15E's parent node. It then inserts another node between the parent node and the polygon nodes for each instrument. This node contains a scale factor that reduces the size of its children by 95% and a translation factor to properly place the gauge or display on the panel. Figure 10 shows the Performer data tree structure after the instrument description was inserted. Because the instrument descriptions are children of the F-

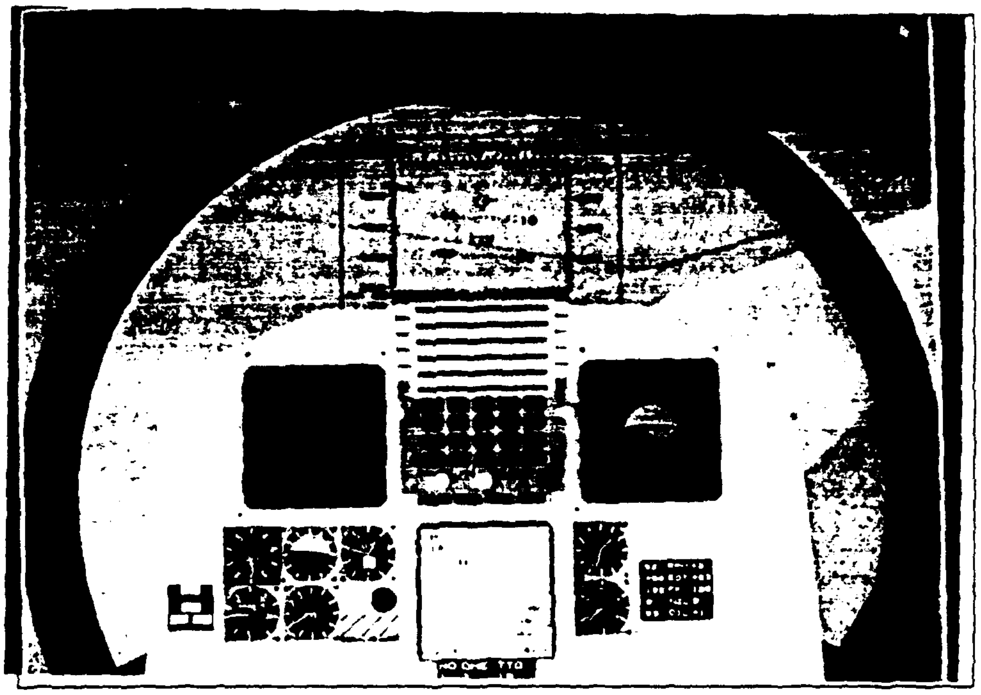

Figure 12. Virtual Cockpit Layout

15E model, the same transformation that updates the F-15E model's position in the simulation also updates their position. This method provides the resolution needed for the gauges, without the overhead of explicitly moving the instruments' polygon descriptions. Once this method was perfected, Lt Matt Breeden created the actual polygon descriptions of the individual instruments, which were then added to the simulation. The layout of an actual F-15E instrument panel is displayed in Figure 11 and the Virtual Cockpit's new instrument panel is shown in Figure 12.

Sound

The interior of any attack aircraft is very noisy. The sounds of the engines, missile lock-on indicators, and missiles firing are an integral part of any simulation. Although adding sound to the Virtual Cockpit was not in the original scope of my thesis effort, the object oriented nature of the Virtual Cockpit made it extremely easy to add sound to the simulation. I utilized a preexisting C++ class that can communicate with a MacIntosh computer via a serial port. When a sound needs to be generated, such as a missile firing, a call is made to a method in this class with the volume, the stereo channel, and an enumerated type defining the sound. The method encodes this information so it can be sent out on the RS-232 serial port. The information is then sent out over a cable connecting the RS-232 port on the SGI workstation to the modem port on the MacIntosh. The software on the MacIntosh interprets the information coming in the modem port, determines the correct sound file to use, and plays the sound on external speakers connected to the MacIntosh. (31)

IV. Weapon Controller

The Weapon Controller acts as the interface between the Virtual Cockpit and all the munitions it carries. It is responsible for initializing weapons, determining when the pilot releases a weapon, and ensuring each active weapon updates its position. The Weapon Controller also allows the user to easily modify the weapon's position on the aircraft, as well as the number and type of weapons carried by the Virtual Cockpit. Table 3 gives a summary of the types, maximum numbers, and approximate weight of the weapons that the Virtual Cockpit can carry (8). The Weapon Controller does not impose a restriction on the number or combination of weapons loaded as long as it does not exceed the limits shown in Table 3; therefore the user must exercise some discretion when configuring the Virtual Cockpit for a sortie.

NAME	TYPE	NUMBER	WEIGHT
MK-82	Unguided convention bomb	18	500 lbs
MK-83	Unguided convention bomb	6	1000 lbs
MK-84	Unguided convention bomb	3	2000 lbs
GBU-12	Paveway II Laser Guided bomb	18	530 lbs
GBU-16	Paveway II Laser Guided bomb	6	1030 lbs
GBU-10	Paveway II Laser Guided bomb	3	2030 lbs
GBU-15	Optically guided bomb	3	2450 lbs
M-61A-1	20mm Cannon	940	2 lbs
AGM-65	Optically guided air to ground missile	6	636 lbs
AIM-7	RADAR guided air to air missile	4	190 lbs
AIM-9	Infrared guided air to air missile	4	503 lbs
AIM-120	RADAR guided air to air missile	4	650 lbs

Table 3. Virtual Cockpit Munitions

THIS
PAGE
IS
MISSING
IN
ORIGINAL
DOCUMENT

41/42

Figure 14. Weapons loaded on the wings

database tree called a RotDCS. The RotDCS is a transformation matrix that describes the position and orientation of a geometric model. The RotDCS of each missile or bomb is set to the load point and is made a child of the aircraft model. Because the bomb or missile RotDCS is a child of the aircraft model, the bombs and missiles remain at the same relative position on the aircraft without explicitly updating their position and orientation. When a weapon is released from the aircraft, its status is set to ACTIVE, the link from the aircraft model to the weapon model is removed, and a new link from the scene node to the weapon model is created. The propagate method in an ACTIVE weapon moves the geometric model through the simulation by updating the RotDCS with a world position and orientation.

The munitions modeled in this simulation do not accurately portray the true characteristics of the actual weapons. However, they do model the essential function of the weapon. For example, my version of a GBU-10 laser guided bomb does not even attempt to follow the flight path of an actual GBU-10, but it will hit the target the pilot has in the cross hairs of the forward looking camera. The reason I made this design decision relates back to the Fidelity versus Realism dilemma. With the resources currently available, accurately modeling the flight path of a bomb or missile would take too much processor time and would cause the frame update rate of the simulation to fall below acceptable limits. However, my design should allow the methods for moving the weapons to be easily changed when more capable computers become available.

M-61A-1 CANNON

The cannon carried by the Virtual Cockpit is modeled after the M-61A-1 20mm cannon (8). The M-61A-1 cannon carried by the F-15E holds 940 rounds of ammunition, and can be used in both air to ground and air to air modes. I based the movement of a cannon round in the simulation on ballistic equations. I ignore aerodynamic forces such as lift and drag and consider only the effect of gravity on the path of the bullet. The ballistic equations require an initial position, initial velocity, the direction of gravity, and a start time. I create a transformation matrix as the first step in calculating the initial position and velocity. This transformation matrix converts coordinates that use the aircraft's center of gravity as the origin into coordinates that use the center of the earth as the origin. I calculate the initial velocity vector by first creating a muzzle velocity vector defined in the aircraft coordinate system. I defined the X component of the muzzle velocity vector to be 1000 m/s and the Y and Z components to be 0 m/s. This moves the round straight out the nose of the aircraft. I multiply the muzzle velocity vector by the transformation matrix and add the aircraft's velocity vector to get the initial velocity. I then find the initial position of the cannon round by taking the position where the round exits the cannon in aircraft coordinates and multiplying that point by the aircraft to world transformation matrix. This point is then translated by the aircraft's world position. The translated point is the initial position of the cannon round in world coordinates.

The direction of gravity is determined when a round is fired by creating a segment from the initial position to the center of the Earth. This segment has a unit vector defining its direction. I multiply each component of the direction unit vector by 9.81 meters/seconds^2, which is the acceleration due to gravity on the Earth's surface. The starting time is simply the time the cannon round was fired.

During each time step in the simulation, the cannon round updates its velocity and then uses the updated velocity to calculate its new position. The new velocity is calculated by multiplying the time elapsed since the last update by the vector describing the acceleration due to gravity and then adding it to the old velocity. The position is updated by taking the new velocity, multiplying it by the elapsed time, and adding it to the last position.

After the cannon round has projected its next position, it must determine if it hit any object in the simulation. The objects in the simulation are kept in an array of pointers by the Simulation Entity Manager, a part of ObjectSim (30). The intersection calculation consists of two separate tests, the Half-Plane Test and the Polygon Intersection Test. These tests are performed in order and if an object fails the Half-Plane Test, then the Polygon Intersection Test is not performed. The Half-Plane Test is the least complex and is performed on every object in the simulation. The test compares the position of an object against the volume inside two overlapping half-planes. The first half-plane is defined by the old position of the cannon round and a normal in the opposite direction to the movement of the round. If the object is in the proper half-plane, i.e. in front of the round, then the object is compared against the second half-plane. The second half-plane is defined by the new position of the round with the normal to the plane in the direction of motion of the round. If an object in the simulation falls in the volume of space interior to these two half-planes, then the more expensive intersection test is performed.

The Polygon Intersection Test compares the segment against the actual polygons that describe an object in the simulation. Only the polygons that face toward the starting pcint of the segment are considered. The culling of back facing polygons is a quick check, and substantially reduces the number of polygons considered in the test. If the segment does hit one of the polygons in the object model, the intersection test function returns the point of intersection. The intersection point is defined in a coordinate system with the origin at the center of the object. Out of the three tests, only the Polygon Intersection Test is absolutely necessary; however, it also takes the most computing time. The other tests are an effort to limit the number of times the Polygon Intersection Test is done.

If the round of ammunition does not intersect any object in the simulation, the altitude of the round is tested against the elevation of the terrain underneath it. If the round's altitude is less than the current elevation, then it impacts the ground, and detonates. If a round of ammunition has not hit another object or the ground after 20 seconds, it automatically detonates. I made this design decision in an attempt to reduce the number of calculations required per frame. After 20 seconds of flight, the round of ammunition has moved 20 Km, well beyond visual range of the pilot, and has missed any target at which the pilot could be shooting. While there is a very small chance the round of ammunition could hit an object, this added bit of realism does not justify the necessary calculations.

A round of ammunition broadcasts two types of Protocol Data Units: Fire and Detonation. The Fire PDU is sent once when the pilot pulls the trigger. A round of ammunition issues a Detonation PDU when it impacts an object, hits the ground, or expires after 20 seconds. If the round hits an object that is controlled over the network by another simulator, the identification for that target and where the target was hit are broadcast as part of the PDU. For local objects such as bridges or buildings, only the location of the detonation is broadcast. These local objects are not controlled by any single simulator, therefore each simulator must determine the effect of a detonation in its own local version of the simulation. This has the potential to cause significant discrepancies between the different simulators, one evaluates a bridge as being destroyed, while another evaluates the same bridge as undamaged. Since single round of ammunition is relatively uninteresting in the distributed simulation, no Entity State PDUs are issued in order to conserve network bandwidth.

Mk-82, Mk-83, and Mk-84 Bombs

The Mk-82, Mk-83, and Mk-84 bombs are the standard high explosive fragmentation bombs used by the US Air Force. The major difference in the actual bombs is their weight, which is 500 pounds, 1000 pounds, and 2000 pounds respectively. Only three differences exist in the way the simulation handles the bombs. The first difference involves how much weight each bomb adds to the Virtual Cockpit when it is loaded. The second difference is the size of the geometric model of the bomb in the simulation. The Mk-84 model is the largest and the Mk-82 model is the smallest. The last difference is the destructive blast radius, again with the Mk-84 being largest. All three types of bombs are unguided and follow a simple ballistic path after being released from the Virtual Cockpit.

The method of updating the position of an unguided bomb is very similar to the method for moving a cannon round. The initial position of the bomb is the load point on the aircraft, which is transformed into a world coordinate position. The initial velocity of the bomb is the velocity of the aircraft at the instant the pilot releases the bomb plus an additional velocity of three meters/second away from the bottom of the aircraft. This additional velocity simulates the push a bomb receives from the ejection rack and ensures the bomb will not hit the airframe when released. The calculations used to find the next position are the same as for the cannon round.

Unlike the cannon rounds, bombs broadcast three types of Protocol Data Units. A Fire PDU is issued when the pilot releases the bomb, and a Detonation PDU is broadcast when the bomb hits the ground and detonates. While the bomb is in flight, it issues an Entity State PDU. The Entity States PDU sends acceleration,

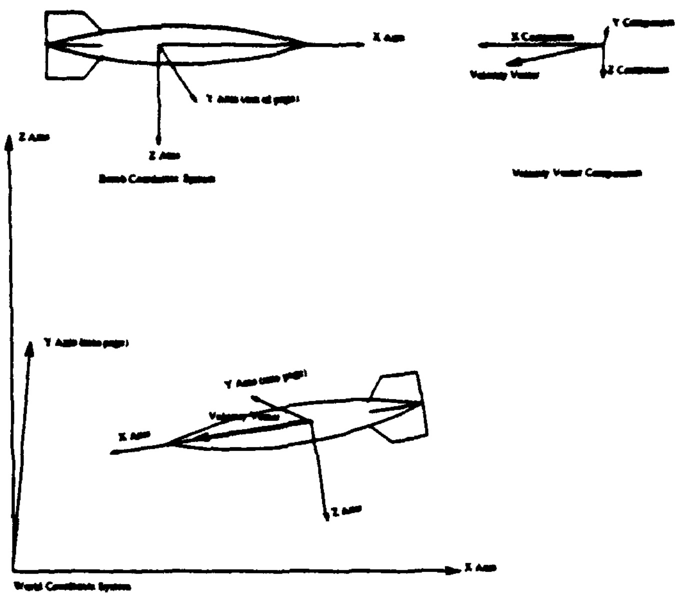

Figure 15. Alignment of Body Axis During Flight

velocity, position, and orientation information out on the network. This information allows other simulators or observers to monitor the bomb as it falls. The bomb must calculate its acceleration, velocity, and position each frame to properly move through the simulation, so broadcasting this information requires no extra effort. Calculating the simulated bomb's correct orientation for the Entity State PDU does require some extra effort. As a real MK-82 falls, aerodynamic forces such as lift and drag act upon it to orient its body to minimize drag while it is falling. However, a bomb dropped by the Virtual Cockpit uses only ballistic equations for its motion and the aerodynamic forces are ignored. Since it is the aerodynamic forces that determine the orientation of a bomb, I created an artificial means of calculating the bomb's orientation. I align the X axis of the geometric model with the velocity vector (Figure 15). I then calculate the bomb's heading by finding the arc tangent of the X component divided by the Y components. The bomb's pitch is the arc tangent of the velocity vector's Z component divided by the length of a 2 dimension vector described by the X and Y components. This method roughly models what actually happens on real bombs and missiles. The tail fins and streamlined shape of the bomb casing attempt to minimize the drag on the bomb as it falls. Drag is minimized by aligning the bomb's long axis with the relative wind. The relative wind is caused by the bomb's movement through the air, and in this case, the movement is represented by the velocity vector. The model orientation method described works well and gives the bomb a natural looking orientation.

I designed the bombs in this simulation to detonate only after they hit the ground. Once the bomb gets its new position, the altitude of the bomb is compared to the elevation of the terrain. If the altitude is less than the elevation, I create a segment starting at the bomb's old position and going toward its new position. The point where this segment intersects the terrain is where the bomb detonates. Once the bomb detonates, a message is sent to the Simulation Entity Manager (30). The Simulation Entity Manager checks all of the objects in the simulation to see if they are within the blast radius of the bomb. If an object that is not controlled by another simulator is in the blast radius, the Simulation Entity Manager sets its status to damaged. Each object that is controlled by another simulator must evaluate the effect of the detonation on itself; thus making it unnecessary to put a target identifier in the Detonation PDU.

GBU-12, GBU-16, and GBU-10 Bombs

Actual GBU-12, GBU-16, and GBU-10 bombs are Mk-82, Mk-83, and Mk-84 bombs with a special guidance unit and movable fins attached. The desired target is illuminated with a laser by the launching aircraft, another aircraft in the area, or ground forces. The guidance package moves the fins so the bomb glides at a high velocity into the target (8). In this simulation, the Virtual Cockpit is the only aircraft that can designate a target. The Virtual Cockpit has a simulated camera underneath the fuselage called the Forward Looking Infrared or FLIR (11). The FLIR has two different modes, the normal mode where it looks at a series of predefined points or the cue mode which lets the pilot manually control the view. In either mode, the point on the ground seen through the cross-hairs of the FLIR is the designated point for the bomb. When the pilot releases the bomb, it evaluates the designated point with respect to its position. If this point lies within range of the bomb, then the bomb follows a vector from its current position straight to that point. This munition is unpowered and therefore the range of a bomb is based solely on its altitude and how far it can glide before hitting the ground. If the cross-hairs of the FLIR do not lie on the ground or the target is out of range when the bomb is released, then the bomb falls in a basic ballistic path. All other aspects of these bombs such as the PDU broadcast and blast radius are similar to the Mk-82, Mk-83, and Mk-84 bombs described above.

GBU-15

An actual GBU-15 bomb is a precision guided 2000 pound bomb. The guidance package for this bomb consists of a television camera and movable fins. The television camera transmits an image from the nose of the bomb back to the attacking aircraft. The pilot or weapon systems officer uses this image to fly the bomb to the target. (8:83) I was able to implement the transmission of the image from the bomb back to the Virtual Cockpit. My version of the GBU-15 uses the FLIR display to show the pilot the bomb camera view. The pilot selects the GBU-15 munition and presses the button mid-way down on the stick handle. From the time the button is pressed until the bomb detonates, the Weapon Controller gets the bomb's world coordinate system position and orientation and saves it in a location the FLIR can access. The FLIR uses the position and orientation information when computing the view shown in the display.

The other aspects of the Virtual Cockpit's version of the GBU-15 are identical to the Mk-84 bomb described earlier. I have not provided a means for the pilot to control the path of the bomb once it is in flight. This capability can be added in the future as part of a project to add a weapon systems officer to the Virtual Cockpit.

Missiles

I chose to implement four different missiles on the Virtual Cockpit. The three air-to-air missiles are the AIM-120 AMRAAM, the AIM-9 Sidewinder, and the AIM-7 Sparrow. The AGM-65 is the single air-to-ground missile. I did not implement any type of aerodynamic model for the movement of these weapons; instead I directly manipulate a missile's velocity vector to guide it towards a target. I took this approach for two reasons. First, the flight dynamics of the AIM-120 and AIM-7 missiles are classified, and the dynamics of the AIM-9 missile are restricted. The use of the actual flight dynamics would severely restrict the distribution of the Virtual Cockpit. Second, the number of calculations required would cause the frame rate to slow down below an acceptable rate.

AIM-120

The AIM-120 Advanced Medium Range Air-to-Air Missile (AMRAAM) is the latest missile to be added to the Air Force and Navy inventory. It has a range of approximately 10 km against receding targets, but the range increases to 50 km against head-on targets (15). The AIM-120's most significant improvement over previous missiles is its launch and leave capability, provided by its own active mono-pulse radar. In short range engagements, the pilot designates a target and the missile radar locks on to it. The pilot then releases the missile, which no longer requires information from the launching aircraft. In long range engagements, the AIM-120 uses an inertial guidance system to maneuver the missile close to the target. After the missile is in the vicinity of the target, the active radar locks on and guides the missile to the target.

I chose to implement the AIM-120 first because of the launch and leave capability. This capability removes any need for the missile to interact with any object other than the target. The pilot of the Virtual Cockpit locks on to the desired target with the RADAR (11) and then releases the missile. The missile keeps a pointer to a data structure that contains the target's position and orientation. The Object Manager constantly updates the position of the target (30). For the first second of flight, the missile moves at 1.2 times the Virtual Cockpit's velocity vector. This maneuver gives a good separation from the aircraft and prevents the missile from hitting the Virtual Cockpit. After the first second, the missile goes into its active tracking and boost phase. During each frame of the simulation, while the missile is in boost phase, it calculates a unit vector from its current position to the target's position. It then scales the unit vector by the magnitude of current velocity vector plus 10 meters per second and uses this as its velocity vector. The boost phase of flight lasts approximately five seconds. At a frame rate of 15 HZ, the acceleration of ten m/s/frame brings the final velocity of the missile to 1700 mph above the velocity of the Virtual Cockpit. This is below the AIM-120's published figure of mach 4 (8), but it is a good approximation. After 40 seconds of flight, I start adding an acceleration due to gravity. The gravity component of the final velocity vector limits the range of the missile and keeps it from flying off to infinity. The missile's orientation is calculated in the same way as a bomb's orientation.

During each frame of the simulation, the missile calculates an azimuth and elevation to the target. If at any time the azimuth or elevation exceeds a 90 degree field of view, the missile breaks lock and no longer pursues the target. While this algorithm does not accurately represent the seeker on a real AIM-120, it does allow another simulator the possibility of evading the missile.

AIM-9

The AIM-9 Sidewinder is an infrared radiation (IR) guided air-to-air missile designed for close-in air-to-air combat. The missile became operational in 1956 and has been through 12 different production models. The missile's seeker unit detects IR sources, such as aircraft engine exhaust, and locks on to the most intense source in its field of view. The range differs for specific models but is typically about 10 miles. The latest versions of the AIM-9 missile have an all aspect angle capability. This means that the missile seeker head is sensitive enough to track a target from all angles. The seeker can detect the heat generated by friction on the target's wings and fuselage as well as heat from the engine exhaust (15). Once the missile is locked on and in-flight, it does not require any inputs from the launching aircraft (8:246).

When the pilot selects the AIM-9 missile in the Virtual Cockpit, it automatically goes into seeker mode. The seeker method compares the position of all the aircraft in the simulation against a cone-shaped frustum. The frustum's sides extend out at 45 degrees to either side of the missile, and its base is 1.6 miles away. If an aircraft is in this volume, the missile then computes its aspect angle. Once locked on and released, the missile checks the target against the frustum each frame. If at any time the target maneuvers outside this frustum, then the missile loses lock and continues to fly in a straight line until it falls. During flight the missile does not consider any other targets or re-establish locks. I chose this design because moving the missile and tracking targets during flight slowed the frame rate down below acceptable levels.

The simulated movement of the AIM-9 is similar to AIM-120 described above. The acceleration of the simulation's AIM-9 is five meters per second per frame for five seconds. This acceleration gives the missile a velocity of 840 mph above the velocity of the Virtual Cockpit at launch. The published speed of the actual AIM-9 missile is approximately maci ..5 above the velocity of the launching aircraft (10:91). The acceleration due to gravity is added in after 20 seconds.

AIM-7

The AIM-7 Sparrow has been through many different versions since the first model reached initial operational capability in 1956. The current model, the AIM-7M, uses a Continuous Wave, Semi-Active Radar Homing to track the target. Like the AIM-120 missile, the Sparrow homes in on radar energy reflected from the target aircraft. In the case of the AIM-120 the radar energy comes from the missile, but the AIM-7 relies on the radar from the launching aircraft. Because the radar homing is continuous wave, the missile requires a constant radar signal return from the target.

The pilot in the launching aircraft must ensure that the radar remains locked on the target aircraft until the missile detonates. Keeping the radar lock on target severely limits the launching aircraft's maneuverability and prevents the pilot from engaging other targets.

To launch the simulated missile, the Virtual Cockpit's pilot designates a target on the RADAR and selects the AIM-7 missile. The missile gets a pointer to the designated target's record structure from the RADAR. After the pilot fires the missile, the simulated AIM-7 missile checks the object currently designated by the RADAR (11) against its target. If the designated target is not the same as the missile's original target, the missile loses its lock on the target. The missile can also lose lock on the target if the target maneuvers outside the missile's seeker frustum. If the missile lock is broken, the missile ceases all course corrections and flies along its current heading until it hits the ground. All other aspects of the missile are similar to the AIM-120.

AGM-65

The actual AGM-65 is an optically guided air to ground missile, more commonly known as the Maverick missile. A television camera in the nose of the missile broadcasts a picture back to the controlling aircraft. The weapons systems officer uses the view from the missile to fly the missile into the target.

The Virtual Cockpit's version of the Maverick missile is unguided but does provide camera images to the pilot via the FLIR. The display in the FLIR (11) is generated in a way similar to the GBU-15, described in a previous section. The motion of the missile is similar to those described above, except the direction of the missile's velocity vector is the same as the aircraft's velocity vector.

V. Head Up Display

The Head-Up Display (HUD) is an essential part of a modern attack aircraft's cockpit instrumentation. A HUD essentially repeats the information shown on head-down performance instruments such as airspeed, altitude, heading, and angle of attack. The HUD also shows other information such as flight path, navigation data, and targeting data depending upon the specific mode of the display. A HUD projects information onto a transparent screen mounted above the instrument panel in front of the pilot. This positioning allows the pilot to monitor flight information and remain aware of what is going on outside the aircraft. The transparent quality of the HUD screen also allows symbols, such as the Target Designation Box, to be overlaid on objects outside the cockpit. The pilot can see objects outside the cockpit through the symbols and know certain information about the objects based on the symbols (9:13).

As stated previously, the HUD was one area of the original Virtual Cockpit that required significant improvement. The previous HUD was designed as a geometric object that existed out of the pilot's view, at the center of gravity of the cockpit. Each possible number position on the HUD was overlaid with the numbers 0 through 9 in a stack. When it came time to render the HUD, the geometric description of each number in the stack was transformed by a 4x4 matrix. The simulation multiplied the numbers and symbols that were visible by a translation and rotation matrix to move them to the proper position, and multiplied all the remaining numbers by an identity matrix. Only one-tenth of the information manipulated was actually displayed, making this design inefficient and slow.

The Virtual Cockpit's HUD design does not exactly match the HUD of an F-15E. Changes were made to the different modes due to the limited user interface in the Virtual Cockpit. Currently, someone flying the Virtual Cockpit uses only the switches available on the HOTAS to interact with all the instrumentation. While

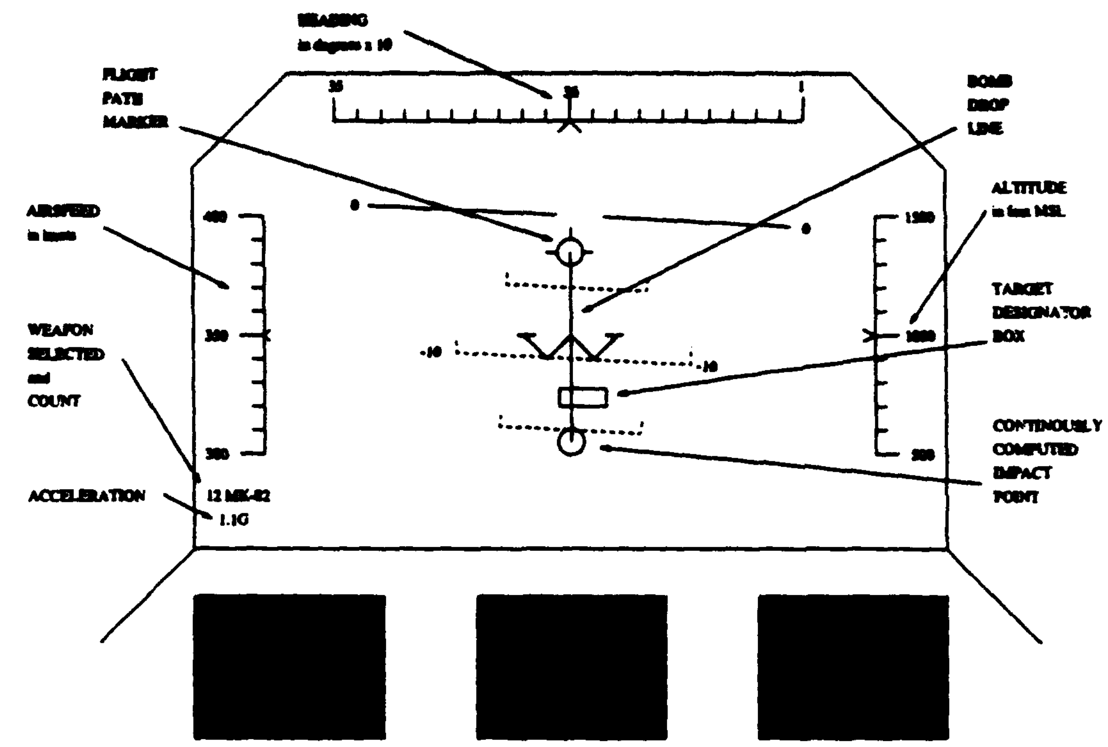

Figure 16. Virtual Cockpit Head Up Display

Capt Erichsen and I made creative use of the HOTAS switches, they cannot fully replace all the switches in a real cockpit. The design is based on information from Air Force Manual 51-37, the F-16 HUD avionics manual and diagrams of an F-15 HUD from a commercial publication (33). Figure 16 shows the Virtual Cockpit HUD in ground attack mode with the different symbols labeled. The Virtual Cockpit HUD requires information from the Weapon Controller and the RADAR objects. The RADAR (11) is responsible for determining the current mode (air to air, air to ground, navigation) of the aircraft system and what target, if any, is currently designated. The Weapon Controller is responsible for reporting the current weapon selected and the count of that type weapon remaining on the aircraft.

The implementation of the Head-Up Display logically divides into two distinct sections. The first section draws the HUD and the second calculates the positions of the targeting and navigation cues. The first section relies on the line drawing functions in the SGI Graphics Library to display the HUD and the flight information.

As explained in the section in Chapter III on multi-processing, the SGI architecture allows only a single processor to have control of the graphics drawing pipeline. Therefore the code to draw the HUD must run on the same processor that draws the polygons in the scene. The draw processor executes the code to draw the HUD through a function call back as soon as it draws the last polygon in the F-15 geometric description. Because the simulation has just drawn the F-15, all the matrix transformations to draw the aircraft model are still on top of the matrix stack. All the coordinates to draw the HUD are passed through these transformations as well. Since the drawing of the scene is the most computationally expensive task in the simulation, a different processor runs the code to calculate the positions of the HUD symbols.

The flight information always shown on the HUD includes the heading, airspeed, altitude, and climb ladder. The heading, airspeed, and altitude indicators are on sliding scales that show trends in the information. For example, as the aircraft climbs, numbers appear at the top of the altitude indicator and move down. The lines in the indicators are created using SGI Graphics Library function calls and are rotated or translated depending on the speed, heading, altitude, and orientation of the aircraft. The text characters that appear on the HUD were more difficult to create. The SGI Graphics Library provides functions for writing text to a graphics window. However, the text characters are not treated the same way as other graphic objects. While the SGI Graphics Library functions place the characters in the right position on the screen, the characters are not scaled or rotated with the rest of the scene. This effect became very noticeable on the climb ladder when the plane rolls inverted or the pilot looks off to one side.

The solution to the text problem required the text to be drawn using the SGI Graphics Library line drawing functions. A character set drawn with line strokes is known as a Vector Font, because the characters are drawn with short lines. I obtained code implementing a two dimensional Vector Font set from the Wright Laboratory

Joint Cockpit Officc. **This code was modified to work in three dimensions and** increase its overall utility.

Cues

The Flight Path Marker, Target Designation Box, Continuously Computed Impact Point, and Gun Sight are four separate cues on the Virtual Cockpit HUD that a pilot uses for targeting and navigation. All four cues relate a point outside the aircraft in the world to a point on the HUD. For each of the cues I will give a brief explanation of its function, and how I calculate the point in the world. After the explanation of the cues, I will explain how a world position is transformed to a point on the HUD.

The Flight Path Marker is a visual representation of the aircraft's velocity vector. The Flight Path Marker shows what the position of the aircraft will be in 20 seconds if the pilot does not modify the velocity vector. It is the velocity vector of the aircraft and not the direction of the aircraft, that will determine where the jet will go. For example, when pulling up out of a steep dive, the nose of the aircraft may point toward the horizon as in level flight, but the momentum of the aircraft will cause it to continue to lose altitude. Of course it is the pilot's input to the aircraft control surfaces that modify the velocity vector, but depending on the specific conditions, there might be a significant amount of lag time between the inputs and the desired direction of the velocity vector. To find the aircraft's position in 20 seconds, the HUD multiplies the X, Y, and Z components of the aircraft's velocity vector from the aeromodel by 20. The resulting value is how far the aircraft would have moved, and this is added to the current position of the aircraft. The 20 second interval was chosen because it places the transformed Flight Path Marker symbol in the center of the HUD when the aircraft is in straight and level flight.

Target Designation Box or TDB is a small rectangle placed around a selected target or navigation point to outline its position in the world. When the target or navigation point is outside the field of view of the HUD, the TDB clamps to the edge of the HUD. For example, if a target is above the Virtual Cockpit at the two o'clock position, the TDB clamps to the upper right corner of the HUD showing the pilot he or she should climb to the right to bring the target into view. The TDB is particularly useful when the target is beyond visual range. The world position of the TDB is the same as the target or navigation point the pilot selected.

The pilot uses the Continuously Computed Impact Point (CCIP) for attacking ground targets with bombs. The CCIP shows the point on the ground where an unguided bomb will impact if the pilot released it at that instant. The CCIP symbol consists of three parts: the Flight Path Marker, the Impact Circle, and the Bomb Drop Line. The Flight Path Marker is the same symbol as described above and represents the aircraft's path on the bomb run. The Impact Circle shows the place on the ground where the bomb will hit. The Bomb Drop Line is simply a line connecting the Flight Path Marker and the Impact Circle. The Flight Path Marker calculation has already been explained. The Impact Point calculation uses the same ballistic equations as those used to update a bomb's position. The ballistic equations are detailed in Chapter 4. The equations iterate with a time increment of 0.1 seconds until the projected path of the bomb intersects the ground. The projected intersection point is used as the world position of the Impact Point.

The pilot uses the Gun Sight to strafe targets with the 20mm cannon. The Gun Sight shows the predicted position of a cannon round fired at any given time. The predicted position is set one second in the future. The Gun Sight uses the same ballistic equations as the 20mm cannon rounds. The equations run once with a time increment of one second.

Viewing Transformation

The position on the HUD of all four of the targeting cues is calculated in a similar manner. I perform a three-dimensional to two-dimensional perspective

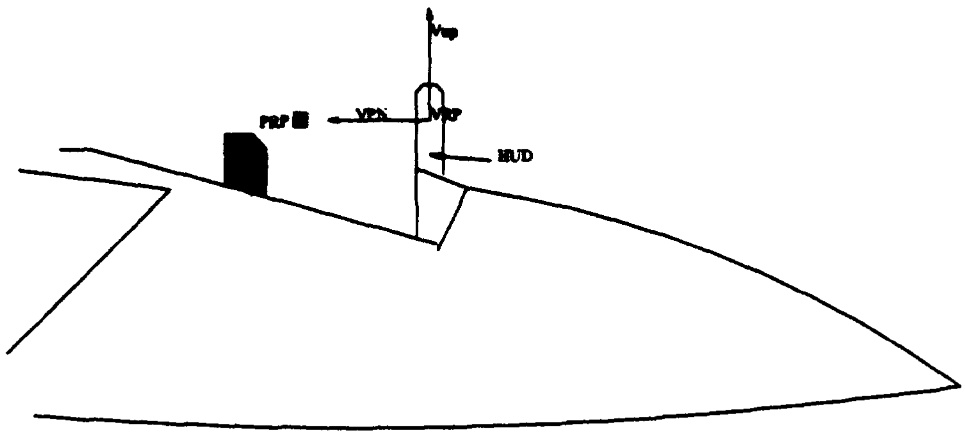

Figure 17. Side View of Virtual Cockpit Head Up Display

viewing transformation on all the world coordinate positions of the targeting and navigation cues. The perspective viewing transformation creates a two-dimensional graphics screen inside the simulation. This type of transformation is described in many computer graphics texts (12). To correctly set up the viewing transformation, the world, aircraft, and HUD coordinate systems must be properly defined. The world and aircraft coordinate systems definitions are described in the section on DIS v2.0 axis systems in Chapter II. Figure 17 illustrates the viewing parameters for the HUD coordinate system and viewing transformation are defined.

Creating the viewing transformation matrix is the first step in converting world coordinates to coordinates on the Head-Up Display. The HUD class initializes this matrix during the simulation start up and does not change the matrix during the simulation. The view reference point (VRP), and two unit vectors, the view plane normal (VPN) and the view up vector (Vup), define the viewing coordinate system. The VRP, VPN, and Vup are described in the aircraft coordinate system. The aircraft coordinate system's origin is at the F-15's center of gravity with the X axis out the aircraft's nose and Y axis out the right wing. The view reference point is the center of the HUD. The view plane normal goes along the X axis of the aircraft toward the pilot's eye point, and the view up vector goes out the top. Once the viewing coordinate system is determined, the viewing volume is defined by setting the perspective reference point (PRP) and the window size, relative to the view coordinate axis. The PRP of the viewing volume is the pilot eye point. The actual size of the HUD defines the size of the viewing window. The front clipping plane equals 0, and the back clipping plane equals 20,000 to ensure that all objects in the view volume will be seen. Once all of these parameters are set, the viewing transformation matrix is built as outlined in *Computer Graphics, Principles and Practice* (12).

A series of transformations is performed on a point in the world coordinate system to find its location on the HUD. First the point in the world coordinate system must be converted to a point in the aircraft coordinate system. This is done by translating the point by the negative aircraft position and then rotating the point by the heading, pitch, and roll of the aircraft. Once in the aircraft coordinate system, the point is multiplied by the viewing transformation matrix described above. A 3D window to 2D viewport transformation converts the point to the aircraft coordinate system so that it can be drawn in the call back routine. In cases where the point lies outside the HUD, the point clamps to the edge for the TDB, Flight Path Marker, and Gun Sight. For the CCIP, the Impact Point is clipped away, and the Bomb Drop Line goes to the edge of the HUD.

VI. Results and Recommendations

Results

At the end of 1992, the AFIT Virtual Cockpit v1.0 was able to fly in a Distributed Interactive Simulation and participated in the Zealous Pursuit Exercise. The frame rate was barely acceptable at approximately seven frames per second and frequently dropped lower depending on what was happening in the simulation. The interior instrumentation, the geometric model used for the airframe of F-15, and models of the other entities were very simple and lacked the required fidelity. The simulation used the SimNet network protocol and assumed a flat earth coordinate system. It was impossible to interact with other entities in the simulation because the Virtual Cockpit did not have any weapons delivery capability, and it was very difficult to find the other entities by only visual means.

During this past year students at AFIT made a significant effort to improve the Virtual Cockpit v2.0 and address the problems mentioned above. My thesis and the work of Mr. Steve Sheasby, Capt Mark Snyder, and Capt Matt Erichsen figured prominently in this endeavor. Due to the complexity of all the different issues, it was impossible for any single person to work in isolation, and in the final analysis it was a team effort. The capabilities of the Virtual Cockpit are shown in Table 4. This table also contains features that are not yet included in this version of Virtual Cockpit but should be considered for the next.

Capt Mark Snyder lead the way in the migration to Performer by creating the ObjectSim framework (30). The class definitions in the ObjectSim structure made porting the existing Virtual Cockpit code to Performer relatively easy. The process of moving the Virtual Cockpit over to ObjectSim took less than three days. The built-in multi-processing support provided by Performer increased the frame rate substantially. The increased frame rate allowed the Virtual Cockpit to use more detailed geometric models for the terrain, instruments, airframe, and other entities

Capability	VC-1.0	VC-2.0	VC-X
Flight Fidelity			
Frames per Second	7	15	15+
Sound		X	X
Viewing Devices:			
CRT	X	X	X
Head-Mounted Display		X	X
miniDART		X	X
Stereo CRT		X	X
Head-Tracking		X	X
Network Compatibility			
SimNet			
Send/Receive Entity State	X		
Send/Receive Fire			
Send/Receive Detonation			
DIS			
Send/Receive Entity State		X	X
Send/Receive Fire		X	X
Send/Receive Detonation		X	X
Send/Receive Emission			X
Send/Receive Lasing			X
Send/Receive Collision			X
Coordinate Systems			
WGS-84		X	X
Flat Earth	X	X	X
Sensors			
RADAR			
Air-to-Air		X	X
Air-to-Ground		X	X
Navigation		X	X
FLIR			
Cue Mode		X	X
Track Mode		X	X
RADAR Warning Receiver			X
ECM/ECCM			X
Avionics			
Inertial Navigation System		X	X
Global Positioning System			X
Tactical Air Navigation			X
Instrumentation			
Simple Texture Map	X		
Polygon Descriptions based on F-15		X	X
Head-Up Display			
Flight Information		X	X
Targeting Information		X	X
CCIP		X	X
Fixed Gun Sight		X	X
Target Designation Box		X	X
Lead Computing Gun Sight			X
Weapons			
Gravity Bombs		X	X
LASER Guided Bombs		X	X
Electro-Optic Guided Bombs			X
Cannon		X	X
RADAR Guided Missile		X	X
IR Guided Missiles		X	X
Electro-Optic Guided Missiles			X
Weapon Systems Officer (Back-Seater)			X

Table 4. Virtual Cockpit Capabilities - Past, Present, and Future

in the simulation. The frame rate over a fairly complex, texture mapped, 25km by 50km terrain (approximately 7400 polygons) with 500 dynamic objects and 200 static objects is 15 to 20 frames per second. The ability to use several different devices to modify a user's view is an additional benefit gained from ObjectSim.

Capt Matt Erichsen developed the RADAR and Forward Looking Infrared (FLIR) Displays for the Virtual Cockpit (11). The RADAR has three modes, (Ground, Air-to-Air, and Navigation) and a range of up to 160 miles. When the RADAR is in Air-to-Air mode, the user can selectively lock on to a target and get detailed information on the target's heading, altitude, speed, closure rate, and aspect angle. The FLIR gives the user an inset view. The viewpoint for the FLIR is on the undercarriage of the fuselage, and the user can change the view direction interactively or cycle through a series of predefined points. Capt Erichsen was also responsible for implementing our solution to the problem of converting from a flat earth coordinates system to the WGS-84 coordinate system.

This thesis outlined the addition of weapons into the Virtual Cockpit, and the steps I took to improve both the simulation's fidelity and frame rate. The Weapon Controller object allows the Virtual Cockpit to carry a wide range of munitions, and the Head-Up Display provides the user with the information needed to deliver weapons on target. The integration of the weapons with the RADAR and FLIR allows delivery of air-to-air missiles and air-to-ground, precision guided munitions. The Virtual Cockpit's frame rate with 500 objects in the simulation stays within the 10 to 15 frames per second, and does work well.

Upgrading the Virtual Cockpit's network interface to work with DIS v2.0 was the task of Mr Steve Sheasby. Moving to the DIS v2.0 protocol required not only reformatting the information being sent but also adding the ability to handle multiple objects generating Entity State PDUs from the same simulation. The weapons also required the ability to broadcast Fire and Detonation PDUs.

The Virtual Cockpit participated in the SIGGRAPH '93 Tomorrow's Reality Gallery (TRG) exhibition (24:214). During the five day exhibition approximately 600 people had the opportunity to fly the Virtual Cockpit. Kaiser Electro-Optics loaned AFIT a 1280x1024 monochrome head-mounted display that allowed participants to fly in a Virtual Environment and participate in a Distributed Interactive Simulation. The Virtual Cockpit was able to interact with the Naval Postgraduate School's simulator, NPSNET, over both a local area network and a T-1 communications line. The terrain database used during TRG was created by the Naval Postgraduate School and populated with a lake, a mountain, a canyon, an airport, and an industrial park. This terrain was dubbed Neyland, in honor of our sponsor at ARPA, Lt Col Dave Neyland. During the exhibition the Virtual Cockpit proved itself in both its ground attack and air superiority roles. The Virtual Cockpit routinely destroyed the industrial production capability of Neyland through the use of daylight precision bombing. Captain Matt Erichsen is the first pilot to become an ace flying the Virtual Cockpit. During the TRG exhibition he was able to destroy over 15 NPSNET aircraft simulators in less than an hour.

Due to hardware configuration problems, the Virtual Cockpit was unable to participate in ARPA's Zen Regard exercise in November of 1993. However, after the exercise I was able to test the Virtual Cockpit against several software systems that act as exercise observers. The tests conducted were intended to provide independent verification of our implementation of the DIS v2.0 network protocol and the WGS-84 coordinate system. The preliminary results of these tests indicate that the Virtual Cockpit is able to send and receive DIS v2.0 PDUs and correctly convert from a local, flat-earth coordinate system to the WGS-84 coordinate system.

Recommendations

The Virtual Cockpit proves that a flight simulator using Virtual Environment technology and Distributed Interactive Simulation protocols can be built. The next logical step is to determine the effectiveness of the simulator. Such a study should be performed by someone knowledgeable in Human Factors, perhaps another AFIT student as part of a thesis. The research should concentrate on the fidelity of the simulation and the effectiveness of the user interface. Above all else, the Virtual Cockpit should not cause the user to develop habit patterns that would be dangerous in an actual F-15.

The C++ classes developed during the 1993 thesis cycle lay the ground work for the building of other DIS simulators. ObjectSim provides the basic frame work in which to build a simulation. The RADAR and Forward Looking Infrared displays should be used as prototypes of sensor packages. These sensor packages can then be installed in a multitude of different simulators such as surface to air missile sites or surveillance satellites. The Weapon Controller can be expanded to handle a wider range of weapons such as rockets, grenades, and projectiles from 105mm down to 9mm. The new weapon objects are then just new classes based on the weapons already developed. Building new simulators based on these C++ classes should be less time consuming, and it should be possible to create a fairly complex simulator as a term project.

Bibliography

1. "Research Directions In Virtual Environments (Special Report)," *ACM Computer Graphics*, p. 156–173 (August 1992).
2. Alessi, Stephen M. "Fidelity in the Design of Instructional Simulations," *Journal of Computer Based Instruction, Volume 15*:p. 40–46 (Spring 1988).
3. Alighieri, Dante. *The Inferno*. New York: The New American Library of World Literature, 1954.
4. Blau, Brian, et al. "Networked Virtual Environments," *ACM SIGGRAPH Proceedings*, p. 157–160 (1992).
5. Bryson, Steve. "Virtual Reality Hardware," *ACM SIGGRAPH Course Notes: Implementing Virtual Reality*, p. 1.3.1–1.3.26 (1993).
6. Chung, J.C., et al. "Exploring Virtual Worlds with Head Mounted Displays." *Visualization and Display Technologies*. pp. 42–52. 1989. SPIE Vol. 1083.
7. Clausewitz, Carl Von. *On War*. Princeton, NJ: Princeton University Press, 1989.
8. Cleave, William Van. *The US War Machine*. New York: Crown Publishers, 1983.
9. Department of the Air Force, Headquarters US Air Force, Washington DC 20330-5000. *AFM 51-37(C3) Flying Training - INSTRUMENT FLYING*, 1979.
10. Duffey, John G. and others. *A Digital Simulation Model for Evaluating System Effectiveness of Infrared Air to Air Missiles*. Technical Report DTIC Report AD-523-236, Air Force Weapons Laboratory, 1972.
11. Erichsen, Matthew N. *Weapon System Sensor Integration for a DIS-Compatible Virtual Cockpit*. MS thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, AFIT/GCS/ENG/93D-7, December 1993.
12. Foley, James D., et al. *Computer Graphics Principles And Practice* (Second Edition). Reading, Massachusetts: Addison-Wesley Publishing Company, 1990.
13. Hanson, Caroline L. "Fiber Optic Helmet Mounted Display: A Cost Effective Approach to Full Visual Flight Simulation." *Proceedings of the Interservice/Industry Training Systems Conference (5th)*. 1983.
14. Institute for Simulation and Training, 12424 Research Parkway, Suite 300, Orlando FL 32826. *Proposed IEEE Standard Draft Standard for Information Technology - Protocols for Distributed Interactive Simulation Applications Version 2.0 Second Draft*, March 1993. Contract Number N61339-91-C-0091.
15. Jane. *Jane's Air Launched Weapons*. Alexandria, VA: Jane's Information Group, 1989.

16. Kunz, Andrea. *A Virtual Environment For Satellite Modeling And Orbital Analysis in a Distributed Interactive Simulation*. MS thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, AFIT/GCS/ENG/93D-14, December 1993.

17. Manno, M. Morris. *Digital Logic and Computer Design*. Englewood Cliffs, New Jersey: Prentice Hall, 1979.

18. McCarty, Dean. *Rendering an Out the Window View for the AFIT Virtual Cockpit*. MS thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, AFIT/GCS/ENG/93M-04, May 1993.

19. McLendon, Patricia. *IRIS Performer Programming Guide*. Silicon Graphics Inc., Mountain View, California, 1992.

20. Miller, Duncan C. "Long Haul Networking of Simulators." *Proceedings of the 10th Interservice/Industry Training Systems Conference*. 1988.

21. Mongold, C. H. *F-15 Aerodynamics Report, Revision*. Technical Report DTIC Report AD-B140-163, McDonnell Aircraft Co., St Louis MO, 1982.

22. Neyland, David. *The Zealous Pursuit Exercise: Overview And Lessons Learned*. Defense Advanced Research Projects Agency, 1993.

23. Orlansky, Jesse and Joseph String. "Reaping the Benefits of Flight Simulation." *Computer Image Generation*, edited by Bruce J. Schachter. pp.191–202. New York: John Wiley and Sons, 1983.

24. Pratt, David. "NPSNET and AFIT HOTAS," *ACM SIGGRAPH Visual Proceedings*, p. 214–215 (August 1993).

25. Rolfe, J.M. and K.J. Staples, editors. *Flight Simulation*. New York, New York: Cambridge University Press, 1986.

26. Rumbaugh, James and others. *Object-Oriented Modeling and Design*. Englewood Cliffs, New Jersey: Prentice Hall, 1991.

27. Scarborough, E. and others. "A Prototype Visual and Audio Display," *Presence, Volume I Number 4*:pp. 459–467 (1992).

28. Schaeffer, Allen. "Performer: Frequently Asked Questions." Usenet post, September 1993.

29. Sheasby, Steven M. *Management Of SIMNET And DIS Entities In Synthetic Environments*. MS thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, AFIT/GCS/ENG/92D-16, December 1992.

30. Snyder, Mark. *OBJECTSIM a Reusable DIS Simulation Framework*. MS thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, AFIT/GCS/ENG/93D-20, December 1993.

31. Soltz, Brian. *Sound Generation Facility*. Air Force Institute of Technology, School of Engineering, Wright-Patterson AFB, Ohio, 1992.

32. Sutherland, Ivan. "The Ultimate Display." *Proceedings of the IFIP Congress*. p. 506–508. 1965.

33. Sweetman, Bill and others. *The Great Book of Modern Warplanes*. New York: Portland House, 1987.

34. Zyda, Michael J. and others. "Flight Simulators for Under $100,000," *IEEE Computer Graphics and Applications*, p. 8:19–27 (January 1988).

Vita

Capt William E. Gerhard Jr. was born in Rockledge, Florida on March 29,1966. He graduated from Plant City High School in 1984. Capt Gerhard attended the United States Air Force Academy from July 1984 to June 1988, and graduated with a Bachelor of Science in Computer Science. His first assignment was to the 7th Communicatious Group, Pentagon, as a Communications Computer Officer. While at the Pentagon, his duties included providing computer support for the Office of the Under Secretary of Defense for Acquisition. From May 1992 to December 1993, he attended the Air Force Institute of Technology, and received the degree of Master of Science in Computer Science. Capt Gerhard is currently assigned to the Arnold Engineering Development Center, Arnold Air Force Base, Tullahoma, TN. He and his wife, Nancy Sue Gerhard, are expecting their first child in January of 1994.

Permanent address: PO BOX 416
Brandon, Florida 33549

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1993	Master's Thesis

4. TITLE AND SUBTITLE

WEAPON SYSTEM INTEGRATION FOR THE AFIT VIRTUAL COCKPIT

5. FUNDING NUMBERS

6. AUTHOR(S)

William E. Gerhard Jr, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/93D-10

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

ARPA/ASTO
3701 North Fairfax Drive
Arlington, Va 22203

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

The Air Force Institute of Technology is continuing research in the Virtual Cockpit. The Virtual Cockpit makes use of high performance graphics workstations, Virtual Environment technology, and Distributed Interactive Simulation network protocols to create a flight simulator based on the capabilities of the McDonnell Douglas F-15E Strike Eagle. The work presented in this thesis focuses on the design and implementation issues for integrating a weapons delivery capability. Weapons simulated include: RADAR and IR guided air-to-air missiles, gravity and precision guided bombs, and a 20mm cannon. Virtual Environment displays used include: color NTSC and monochrome high resolution helmet mounted displays employing a Polhemus Fastrack sensor, and a display using five separate BARCO projectors simulataneously. The Target graphics system was a four processor, SGI Onyx workstation with a Reality Engine graphics pipeline. Graphics rendering was accomplished with an AFIT developed object oriented simulation software package based on the SGI Performer 1.2 application development environment.

14. SUBJECT TERMS

Simulation, Flight Simulators, Distributed Interactive Simulation, Synthetic Environments, Head-Up Display, Missile Simulation, Computer Graphics

15. NUMBER OF PAGES

78

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL